Human Robot Interaction
Dialog Management *

Sumire Honda (SH)
University of Potsdam
sumire.honda@uni-potsdam.de

Jatin Karthik Tripathy (JKT)
University of Potsdam
jatin.karthik.tripathy@uni-potsdam.de

Clara Wicharz (CW)
University of Potsdam
clara.wicharz@uni-potsdam.de

Sebastian Wilharm (SW)
University of Potsdam
sebastian.wilharm@uni-potsdam.de

Abstract (JKT)

The pentomino puzzle game is an example of a standard Human-Robot-Interaction where the human instructs the robot to hand over pieces which the human is then responsible for arranging into a figure. In our project, the robot receives instruction from the human in three different modes, the Language Team, the Language & Vision Team and the Gesture Team. Our task was to build a Dialog Manager that is capable of taking input from the three teams, make a decision and then pass on the decision to the Motion team, who are responsible for the actual robot movement. To this end, we implemented three different approaches - a rule-based decision tree, a Deep Learning method and a Reinforcement Learning method. Since there was no real-world data, we also built a synthetic dataset that could be used for training and testing the different approaches. All three of our approaches performed quite well on the synthetic data, with around a 95% accuracy score. While we cannot say for certain which is the best method, each approach as well their advantages and disadvantages will be discussed in this paper.

*https://gitup.uni-potsdam.de/pfennigschmi/hri-pentomino/-/tree/group-B-dialog-manager/

Contents

1 Introduction (SH)

In the class project, we create a robot arm dialogue system to play the pentomino puzzle game with a human. It is a turn-based game in which the human expresses to the robot which piece they would like. The robot then hands that piece to the human, who then places it on their side of the table to assemble a predetermined figure.

Our group task is working with the Retico framework, setting up the voice input, communicating with ROS component and designing the Multimodal Dialogue Management (DM). To this end, we developed three different models that use the input from the three previous teams to make a decision.

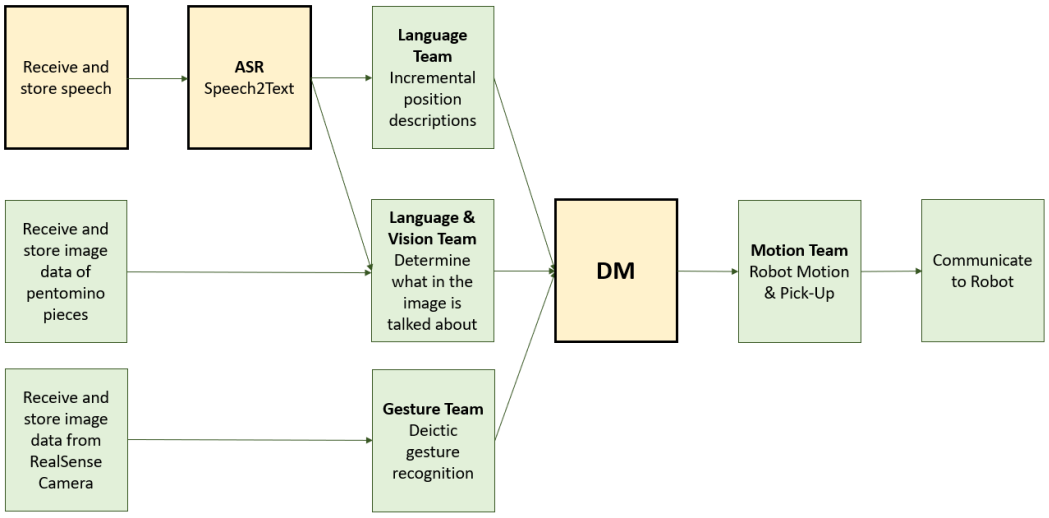

Figure 1: The role of DM and ASR in our class task

2 Multimodal Dialogue Management (SH)

In our class project, the role of multimodal DM is to make a decision on what action the robot should take given the multimodal information by a human;

- Verbal Info: on incremental position description (**Language Team**)
- Verbal + Visual Info: on determining which piece is talked about by Language Team (**Language & Vision Team**)
- Visual Info: on deictic gesture recognition (**Gesture Team**)

The dialogue manager needs to manage consistent or conflicting information output by each of the three teams, and decide which coordinate to move to pick up the desired pentomino piece or express its uncertainty to get clearer instructions from the human (Figure 1). Based on the decision, we will give the information on the coordinate of the pentomino piece the robot arm should pick up and the flag indicating the action movement to the Motion Team. Motion Team is on the right side of DM in Figure 1.

In general, a Dialogue Manager (DM) is a component of a dialogue system, which is responsible for the transmission of information among participants in human-machine interaction (Merdivan et al., 2019). As in Figure 2, DM consists of two parts. One is Dialogue State Tracking (DST), and the other is Dialogue Response Selector (DRS). DST tracks the current status of the dialogue, and DRS decides what action the robot arm should take given a certain state of the dialogue that is tracked with the DST (Merdivan et al., 2019). To enable DRS to decide the action, we need to train it beforehand, which is called Policy Learning (Ni et al., 2022).

Specifically for our task, we work with Retico framework for the entire DM, and for DRS, we use the three models: rule-based baseline model, Deep Learning, and Reinforcement Learning. Our Deep Learning model and Reinforcement Learning model are trained on the process of Policy Learning to train DRS. We will explain about each model in further sections.

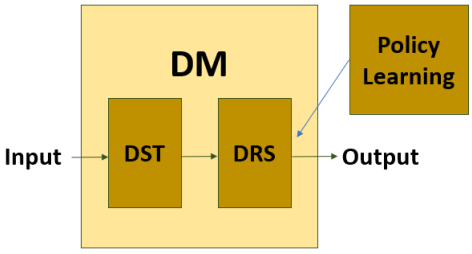

Figure 2: DM structure

2.1 Input and Output format of DM

Our role as DM is to determine what action the robot arm should take given the three multimodal information from the three different task teams and order the action to Motion Team. Thus, if we think of our DM as a function, we need to get three different teams' output as our DM's input, and we hand over the decided action as an output.

For the input of our DM, we asked all of the three teams to have Instruction Confidence and Flag. The example input is shown in Table 1. Instruction Confidence is a float number ranging from 0 to 1, which indicates if the instruction by a human was given or not. When it is given with 100 % confidence, it is 1, and the flag is an integer label that indicates how the robot arm should be moved. The first three flags are in common among the three task teams; label 0 is uncertainty, 1 is absolute movement, and 2 is relative movement. Absolute movement is flagged when we choose the Language & Vision or Gesture Team's input, and the coordinate represents an absolute position on the table. Relative movement is flagged when we choose Language Team's input, and the coordinate represents a relative movement vector.

To Language Team, we asked to express the pentomino coordinate with the relative vector from the current position. The position is described with three-dimensional vectors of the x, y and z axis. Since Language Team has an NLU module that transfers human utterance into robot arm movement, we asked them to express additional flags such as stop, grab, release, reset, and reverse with the integer flag label 3 to 7.

For the Language & Vision Team and Gesture Team, we asked them to express the Coordinate ID and the Coordinate Confidence. Coordinate ID explains the pentomino piece's position with the coordinate system that each team created. The ID is an integer ranging from 0 to $n - 1$, where n is

the number of the pentomino pieces. Coordinate Confidence is a confidence value of their Coordinate ID, which is a decimal value ranging from 0 to 1. By having the Coordinate Confidence, DRS in DM can compare the multimodal coordinates based on how reliable the team's information is and give the information to the Motion Team. For example, in Table 1, we can see that the Language & Vision Team and the Gesture Team agree with the Coordinate ID and the flag 1 for Absolute Coordinate. This information will be likely to be sent to the Motion Team, as we described in Table 2.

Table 1: DM Input Example

From Team..	Instruct Conf	Coord ID	Coord Conf	Relative Vector	Flag
Language	0.3	-	-	[$\vec{\Delta}$x, $\vec{\Delta}$y, $\vec{\Delta}$z]	3
Language & Vision	0.9	1	0.85	-	1
Gesture	0.6	1	0.5	-	1

Instruct = Instruction, Conf = Confidence, Coord = Coordinate, Move = Movement

As for the output of our DM, the example is shown in table 2. We should output the Absolute Coordinate or Relative coordinate, Absolute Coordinate Confidence or Relative Coordinate Confidence, and Flag to the Motion Team, so that the robot arm takes the next action or expresses the uncertainty to communicate with the human. For example, the agreed Coordinate ID in Table 1, which was transformed considering each group's Absolute Coordinate system and the flag are reflected in Table 2. The Coordinate Confidence should be predicted by the DRS based on each Team's confidence value. Absolute / Relative Coordinate will consist of integers or decimal numbers, and Absolute / Relative Coordinate Confidence is a decimal number ranging from 0 to 1. The flag is an integer ranging from 0 to 7, as in figure 5.

Table 2: DM Output Example

To Team...	Absolute / Relative Coord	Absolute / Relative Coord Conf	Flag
Motion	[785.9, 326.4, 10]	0.675	1

3 Incremental Processing (JKT)

By the virtue of how humans generally communicate with each other, most conversations that occur between humans tend to be incremental in nature. Incremental processing can be simply defined as starting the processing of the information received before the entire message is received. In humans, we automatically perform incremental processing during a conversation or discourse in various different ways such as starting to understand utterances based on facial cues or eye movement. Even the fact that humans can understand utterances that only make sense when accompanied by a previous sentence is a form of conversation when incremental processing is required for mutual understanding.

Given that human interactions are so in tune with the usage of incremental processing, any Human-Robot interaction would also require the robot to implement some form of incremental processing to

feel natural. In our specific project, the need for incremental processing becomes far more pressing as a bulk of the communication with the robot itself is conducted with speech. Unlike written text, processing incrementally becomes far more necessary for two main reasons.

The first is the fact that in the text the data is already set in stone and can be processed fairly quickly while on the other hand quite a lot of information is encoded in forms that are not lexical in nature, and sometimes hesitance could also be an indication for a certain object (Arnold et al., 2007). The second reason while being fair and simpler is deceptively harder to attempt and solve, the fact that humans generally tend not to wait for the other person to completely stop speaking before performing an action makes them expect the same from a robot that they interact with.

Both of these two reasons together however do inherently also pose another issue that is not usually present when considering an input in the form of text. While humans can be considered great communicators, in fact, it would not be a stretch to state they are the best communicators that are currently present on Earth, they make mistakes. Especially when placed in an environment such as the one that is tackled by our project - a game.

Incremental processing will allow for a much quicker change in decisions for the robot itself when compared to only taking an action after the entire input has been received. Beyond this, incremental processing also has various other advantages that do not pertain directly to how natural the human-robot interaction feels. Understanding based on partial speech recognition (Sagae et al., 2009), determining when to respond during ongoing utterances (DeVault et al., 2009), training actor policies for quick task-based dialogue (Paetzel et al., 2015), continuous understanding and acting (Stoness et al., 2005), incremental repair detection (Hough and Purver, 2014), or incremental reference resolution (Schlangen et al., 2009) are a few examples.

However, while incremental systems feel more natural during human-robot interaction, they pose several challenges that do not occur while building a non-incremental system. Köhn (2018) outlines a few of the issues that may arise while building an incremental system.

The size into which the input and output are separated depends on the granularity. Every system can be thought of as incremental if the granularity is coarse enough. When processing a paragraph where the basic units are sentences, a typical syntax parser processes the paragraph incrementally rather than non-incrementally within each sentence. In our project, in particular, the granularity of the input data itself is predetermined for the Language Team and the Language & Vision Team since text to speech is done using Google ASR. As for the Gesture Team, the gestures themselves can not be broken down further, and thus can also be thought to be the most granular form of that input.

Monotonicity is another aspect of an incremental system which we did not have to concern ourselves with, much like granularity. As previously mentioned, the inputs to the Language Team and the Language & Vision Teams come from Google ASR, by default our Dialog Manager needed to work with non-monotonic input, i.e. new input the Dialog Manager could retract previous messages.

The alignment of the created output to the components of the input that provided support for this output is referred to as grounding. By using grounding, we can determine whether a portion of the output can be logically produced with only a partial input. Some tasks, such as sequence labelling tasks where each input element is given a label, make this alignment evident in both test and training data. In our project, grounding was one of the main challenges that we had to overcome as it was incredibly difficult to judge which inputs from each team were co-related to each other; and thus difficult to judge how and when the Dialog Manager should take decisions.

Much like grounding, timeliness is also closely related to the alignment of the input and the output. But unlike grounding, timeliness is more concerned about when the output should even be generated rather than being concerned about the alignment in the input. While there is no clear answer to this, in general, to create a system that feels the most natural, the output should be generated as quickly as possible when an input is received. That said since our project is non-monotonic, immediately processing the inputs can have the issue of making the system feel very "jerky" if a new input revokes the previous output.

While our system also faced the same hurdles, several of the decisions were made simpler since a bulk of the incremental system that we built was based on other frameworks like Retico (Schlangen and Skantze, 2009) and Google ASR both of which will be discussed in the following sections. There were some issues that we had to work around, and the solutions that we had come to use the mentioned frameworks will also be discussed in the following sections.

4 Automatic Speech Recognition (SH)

In our dialogue system, Automatic Speech Recognition (ASR) plays the role of being responsible for converting users' utterances into text. In Figure 1, each square stands for a different team's modules and ASR is located in the Speech2Text module. The text output of the ASR Module, given the human utterance, will be the input of the Language Team. In related Human Robot Interaction (HRI) projects, it is common to use the existing ASR API. For example, Google API, Microsoft API, CMU PocketSphinx etc., are widely used engines because of their significant improvement using Deep Learning technologies in recent years (Berg and Lu, 2020).

For our project, we chose Google Cloud Speech to Text (Google ASR) because of two reasons; the availability to work with the existing main DST Retico and its high performance (Michael, 2020). As for the former reason, Retico provides the incremental module for running Google Cloud ASR in the environment, so the compatibility makes us to adapt Google ASR. For the latter reason, according to Li et al. (2017), the word error rate of Google ASR has reached to overall 4.9 % already in 2017, which is adapted with multichannel Google Home specific data. The performance resulted from the experiments on about 18,000 hours of noisy training data consisting of 22 million English utterances. The varying degrees of noise are artificially added using a room simulator. Since we assume our dialogue system will also be carried in English with a reasonable extent of noise and similar daily

conversational vocabularies, we expect the Google ASR to retain a reasonable level of performance on our task with DM too.

4.1 ASR Testing and its limitation

To check the performance of the Google ASR, we tried to test the system and output the text from the test utterances. In this test, we tried to make the environment as silent as possible. The speaker made utterances in a completely silent room at home and tried to make utterances that would likely be made when playing the pentomino game with the robot arm. We assume the game is not always played by the native English speaker, so the test speaker's mother tongue is not English either.

Below, we attach the example output of the Google ASR test utterances. In each test example 1 to 8, the first line shows the speaker's actual utterance, and the second line outputs the Google ASR's final output after the utterance is over. Inside the tuple, the first string shows the understood utterance, the second number shows the confidence value of the output, and the last Boolean flag shows if the sentence has been fully parsed.

Through this experiment, we found two things. First, ASR struggles to figure out where the word boundaries are. In example 1, ASR failed to segment the word "right angle piece", and it wrongly separated the word "angle" into "and" failing to recognise the consonant "g" of the word "angle", and it wrongly connects the sound "gle" in "angle" with "piece" and it became "groupies", mixing the consonant "l" in "gle" and "r" in "groupies".

Second, some word which indicates the pentomino pieces is not detected correctly. For instance, in the example 3, 4, and 5, the pentomino piece names "the green T", "the yellow C", and "the orange L" based on their shape are not recognised as an alphabet; instead, they are understood as "the green tea", "the Yellow Sea", and "Oragel". These outputs show the difficulty of ASR's recognition of homophones. The piece name "T", "C", and "L" are difficult to recognise unless the model knows these alphabet sounds indicate the pieces' names. On the other hand, example 2, 6, 7 and 8, which does not include alphabetical piece names, are successfully recognised by the ASR.

1. `Please give me the brown right angle piece`
`[('please give me the brown right and groupies', 0.8274574875831604, True)]`

2. `Give me the blue stick`
`[('give me the blue stick', 0.0, 0.9630869626998901, True)]`

3. `The green T`
`[('the green tea', 0.7833167910575867, True)]`

4. `Can you give me the yellow C at the bottom right`
`[('can you give me the Yellow Sea at the bottom right', 0.953690767288208, True)]`

5. Hand me the orange L on the left

[('hand me the Orajel on the left', 0.763403058052063, True)]

6. Give me the yellow piece

[('give me the yellow piece', 0.8970740437507629, True)]

7. Give me the piece in the center

[('give me the piece in the center', 0.9190965294837952, True)]

8. The blue piece on the left

[('the blue piece on the left', 0.9455029964447021, True)]

From this test, we found that the Google ASR does not perform perfectly in the given situation, depending on the vocabulary we use. For future work, we should try to find a better ASR system or train the ASR model with a certain domain's vocabulary. However, it is not realistic considering the availability of the compatibility with the DM system and its limited access to the dataset. As a realistic and simplified solution for this project, we propose that the speaker should not use alphabetical piece names in the experiment.

5 Retico (SW)

Retico is an open-source modular framework for incremental processing in dialogue systems written in python, which facilitates research on incremental dialogue. It provides a number of already implemented modules and is comparatively easy to extend with custom functionality that is more adjusted to the required task. A few extensions already exist that provide specific functionality like automatic speech recognition (`retico-asr`) or text-to-speech conversion (`retico-tts`). (Michael and Möller, 2019)

5.1 Incremental Units

Incremental units (IU) are the packages of the Retico world. They contain the information that is being transmitted along with information what created them (`creator`) and when (`created_at`) and links to both their predecessors of the same type (`previous_iu`) as well as their predecessors in terms of what information was used to build them (`grounded_in`). They are freely modifiable python objects and as such any information can be stored in their class members and thus conveyed to later processing steps. Incremental units are transmitted in update messages together with an update type of which there are three commonly used ones. ADD is used to add information to the current state, REVOKE is used to revoke a previous incremental unit, i.e. remove information from the current state and finally COMMIT is used to mark an incremental unit as final and not to be changed. (Michael and Möller, 2019; Michael, 2020)

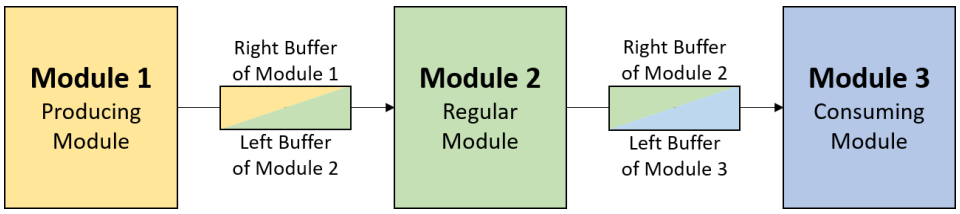

Figure 3: Illustration of module connections via shared bufffers

5.2 Modules

The key building blocks that make up a Retico program are the so-called modules. They represent the stations in which incremental units are processed. When creating a custom incremental module, the list of possible input IUs as well as the output IU need to be defined. Besides regular modules that both take in and return an incremental unit, Retico distinguishes between three special types of modules: consuming modules, producing modules and trigger modules. Consuming modules do take an input IU but do not feed any incremental units into the Retico system and as such represent the end of a pipeline, producing modules in turn do not take an input IU but do generate an output IU constantly, while trigger modules are producing modules that only generate an output when a trigger function is called. The framework provides a setup function, which is run once when the module is initialized and a shutdown function, which is run just before the module is stopped. This can be used to initialize of tear down more complex processing functions, e.g. loading the model weights of a neural network.

The processing pipeline of an incremental module consists of a left buffer per preceding module, the `process_update` function and a right buffer per following module. Whenever an input IU is added to a left buffer, it is run through the process function and the resulting output IU is added to all right buffers. The subscribing i.e. connecting of two modules is done by having one module's right buffer as the next module's left buffer. This is illustrated in Figure 3. Since each module is run in a separate thread, making the entire system work in parallel, a subclass of the thread-safe dequeue class, a so-called incremental queue, is used to guarantee consistency. (Michael and Möller, 2019; Michael, 2020)

5.3 Implementation

Corresponding to each of the project teams, we created five custom modules and incremental units and linked them all up together with two preexisting Retico modules. The resulting module graph can be seen in Figure 4. The processing inside these modules is task of the corresponding teams and will be further discussed in Section 10. All modules are started at the beginning of the program and are closed again at the very end.

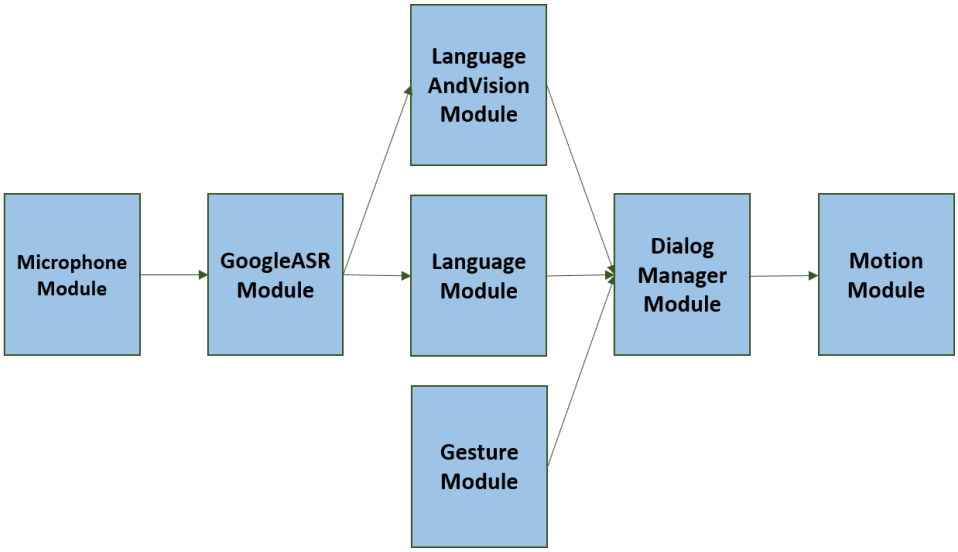

Figure 4: Retico module structure

5.3.1 Automatic Speech Recognition

As discussed earlier in Section 4, we decided to use the Google ASR module already implemented as a Retico extension. To this end, we initialized both a microphone module and the Google ASR module with standard parameters and connected them. The microphone module comes already with Retico and is an example of a producing module as it does not take incremental input but produces audio IUs. These audio IUs are then sent to the ASR module, which sends the recorded audio to the Google Speech API and receives back its transcription, which is then packaged as a speech recognition IU.

5.3.2 Language Only

To represent the Language Team in our system, we created both a Language Module and a corresponding Language IU class. In accordance with the communicated format, we set the IU to contain an instruction confidence, a three-dimensional vector to represent the intended movement and a flag for custom signals. The Language Module takes in a speech recognition IU produced by the ASR module and outputs a Language IU.

5.3.3 Language & Vision

The Language & Vision Module and incremental unit are very similar to the Language ones, the difference being in the payload of the IU. Since the Language & Vision Team is generating a probability for each pentomino piece, besides the instruction confidence, we here store a dictionary of pentomino pieces with their corresponding probabilities. The module again takes in a speech recognition IU and produces a Language & Vision IU. The Language & Vision Team also handles

input from the overhead camera but this is not going through Retico and as such not represented by an incremental unit.

5.3.4 Gesture

The Gesture IU looks the same as the Language & Vision IU as the agreed upon input format is the same, but the module this time does not have an input IU since its inputs come only from the two cameras and not from another Retico module. Therefore, the module is implemented as a trigger module, which is being triggered in regular intervals.

5.3.5 Dialog Manager

The Dialog Manager module and incremental unit are the heart of our system. Here we take and store the outputs from Language & Vision, Language and Gesture Modules, make the decision what action to perform and send that decision to the Motion Module.

```
class Flag(enum.Enum):
    UNCERTAINTY = 0
    ABSOLUTE_MOVEMENT = 1
    RELATIVE_MOVEMENT = 2
    STOP = 3
    GRAB = 4
    RELEASE = 5
    RESET = 6
    REVERSE = 7
```

Figure 5: Possible values of the flag

5.3.5.1 Module, Incremental Unit & flag

The Dialog Manager module has multiple possible inputs: Language & Vision IU, Language IU and Gesture IU. This brings with it the challenge of grouping asynchronous inputs which will be discussed in Section 5.3.5.3. The output of the module is a Dialog Manager IU.

The incremental unit stores the confidence in the decision, the coordinate that was decided on and a special flag. For the special flag we thought of three cases: UNCERTAINTY, the input is inconclusive and the robot should express an uncertainty gesture, ABSOLUTE_MOVEMENT, the deciding input comes from Language & Vision or Gesture Team and the coordinate represents an absolute position on the table or RELATIVE_MOVEMENT, the deciding input comes from the Language Team and the coordinate represents a relative movement vector. To these three we added the five special flags that the Language Team came up with: STOP, GRAB, RELEASE, RESET and REVERSE. The corresponding flag values can be seen in Figure 5. This could be freely extended with more options, as long as those are coordinated with the Motion Team.

5.3.5.2 Memory

Since the `process_update` function in Retico is only ever called with IUs of one type at a time, it is necessary to store the IUs to make a decision that takes all three input teams into account. To this end we check every incoming IU for its type and store it in a corresponding class variable. Storing more than one IU per team is not necessary as the IUs themselves include a reference to their predecessor so the last one enables us to access all previous ones from the same team. This also allows us to handle the special incremental update type REVOKE by setting that team's stored IU to be the `previous_iu` of the revoked one thus rolling back the history to that point.

5.3.5.3 Asynchronicity

As previously mentioned, incremental units from the other modules do not come grouped together but one by one, but can often refer to the same instruction given by the human player. Figuring out therefore which incremental units belong together is crucially important. If the human points at a piece and at the same time describes it, this combination will greatly help identifying the correct piece to pick up. When in contrast two separate instructions are erroneously processed as describing the same request, the result will likely be wrong.

To solve this issue, we decided to use the timing of the incoming instruction. For this purpose, we use the second type of memory that an IU brings with it, the reference to the IU that was used to create this one. Since both Language & Vision IU and Language IU are created based on a speech recognition IU, finding the two IUs that belong together is simple by comparing the IUs they were grounded on. For Gesture Team this is not quite as trivial so we instead use the time when the Gesture IU was created. We set a timing threshold within which two IUs are considered to be referring to the same instruction.

The risk now is that the corresponding IU from another team might still be coming in the future, so we devised an approach in which we always look at the oldest of the three current IUs and make sure it is at least as old as the timing threshold. We then compare the creation time of the oldest IU with those of the other two IUs. If they were created within the threshold, we process them together. If an IU of another team is too old, we instead use an empty IU with zeroed out confidences.

Figure 6 shows an example of timings of IUs coming into the Dialog Manager. We see the three teams represented in different colors and the numbers in the boxes indicate the time that is used to determine when the instruction was given. We will show the result of processing these IUs with a threshold of 1.

The three IUs currently stored in the Dialog Manager module will be the rightmost three, the oldest of which is the Language & Vision IU that was created at time step 8. As we are at time step 15 now, this IU is old enough to be processed. Comparison with the last Language IU from time step 12 shows that this is newer and therefore we instead look at its predecessor. Since that one also has a time step of 8, this is to be grouped with our Language & Vision IU. Applying the same process to the Gesture IU again shows the last IU to be newer so we again proceed to the predecessor. However,

since the difference between this and our comparison IU is 2 and our threshold is 1, we instead use an empty dummy IU for the Gesture Team. The resulting IU set for inference is therefore the Language & Vision IU from time step 8, the Language IU from time step 8 and an empty Gesture IU.

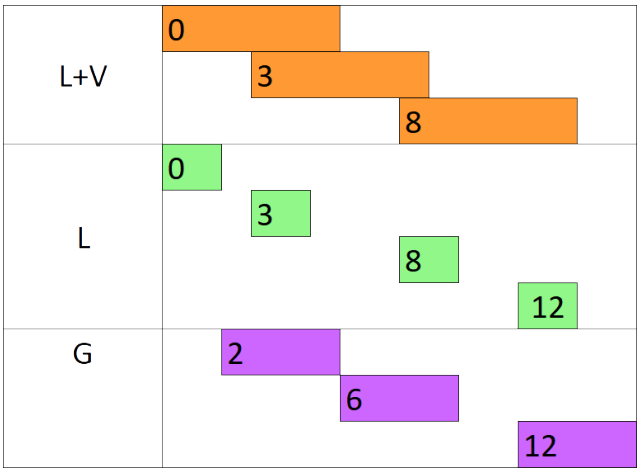

Figure 6: Example timing of incoming incremental units

5.3.5.4 Inference

The inference of the decision happens with one of our three models, the decision tree, the Deep Learning model or the Reinforcement Learning model. The desired model is set in the constructor of the Dialog Manager module and if needed the weights are loaded in the setup function. After determining the set of corresponding IUs, the chosen model is applied.

The output is then translated into either an IU with the uncertainty flag, an IU with the inputs from Language Team or an IU for the corresponding determined piece coordinates. If the current input is a revoked IU, the new decision is compared with the previous one to determine if a new instruction needs to be conveyed to the Motion Team. If the instruction changed, the previous instruction is revoked and the new one added, if the instruction stayed the same, no output IU is generated, since the correct action is already queued.

5.3.6 Motion

The Motion Module is the end of our pipeline and as such there was no need to create a Motion IU. The module takes in a Dialog Manager IU, which contains the decision the Dialog Manager made and is then responsible for communicating that decision with the Motion Team. Since there is no output IU, this is implemented as a consuming module.

5.3.7 Future Work

It is not entirely clear at this point to what extent our Retico code would need to be adapted to facilitate multi-round processing. There might be the need to store hidden or resulting states in class variables to have them available when the next instruction comes in, but the majority of the required

changes should be within the models themselves. Commit flags could be used to mark IUs as ending a sequence and thus enabling tracing back the predecessor references to the beginning of the current sequence using the built-in memory.

6 Dataset (CW)

While a rule-based DRS only requires data to be evaluated on, Deep Learning (DL) and Reinforcement Learning (RL) based approaches require training data. DL requires supervised data, RL requires a weak reward signal for training (Ni et al., 2022). Since RL does not only learn to predict one action, but a sequence of actions, it requires more data.

Ideally data is generated naturally to minimizes a covariate shift. However the extent of human labor is often an unattainable requirement. Motivated by the scarcity of naturally generated data, Zhang et al. (2019) proposed budget-conscious scheduling (BSC) for reinforcement learning. The number of human interaction is treated as budget. With a probability scheduler human labor is allocated during training. Furthermore, a controller determines when to train on natural, when on simulated interaction. With only a small fixed amount of user interactions training could be improved for task-oriented dialogue agents over training with mere synthetic data (Zhang et al., 2019).

In our case, it was unfeasible to train only on naturally generated data. Also, our DS was expected to be up and running only by the end of the semester. Therefore we did not even expect to attain small naturally generated data to complement RL training on synthetic data with approaches such as the BSC. Therefore we based all our approaches merely on synthetic data. For the generation several simplifications were made.

6.1 Simplifications

To align with agreements regarding each module's output and to quickly create a MVP we made the following simplifying assumptions about natural data:

Incremental Position Descriptions. The Language Team needs to provide the DM with an Instruction Confidence and has no further constraints regarding their output. Since incremental position descriptions are supposed to be prioritized over other instructions, the DM will merely base its decision on the Language Team's Instruction Confidence. In case of high enough confidence the DM simply forwards the respective content. Therefore we did not simulate the content of the Language Team.

Continuous Output. For simplification we assume that previous groups always output, even if no human instruction of the respective modus was given. Therefore the DRS always makes a decision based on three streams of non-zero input. This is an oversimplification, since the ASR module only outputs when it is confident enough that speech was presented. Therefore the Language & Vision Team and Language Team do not output, if the ASR module is not outputting. Also, while the camera

always outputs, the Gesture Team only sends output when it is confident enough in the instruction interpretation.

Single-Turn Dialogue. The human-robot interaction is limited to one turn: the human is presenting an instruction, the robot is reacting to it. In this case training data for RL is not different from supervised data for a DL model. This is an oversimplification defying the purpose of RL. However, it allowed us to build a competitively performing MVP that can be further developed to a multi-turn scenario as described in Section 8.3.6.

Absence of Contradictory Instructions. The human does not present conflicting instructions on different communication modi.

Instruction Probability. We assume it to be equally probable that an instruction was given and that an instruction was not given.

Puzzle Piece Probability. All puzzle pieces are equally likely to be drawn. This is an oversimplification, since this is only the case for the first puzzle pieces that is supposed to be picked up when playing the Pentomino game.

6.2 Properties of Naturally Generated Data

Given the agreements with other teams and the simplifications stated above, we can reduce the previous team's output that enters the DM to Instruction Confidences and Coordinate Confidences. Instruction Confidence values are produced by all teams, Coordinate Confidence values only by the Language & Vision and Gesture Team. Both types of confidence are based on the existence or absence of a human instruction. Ideally, previous groups would assign an Instruction Confidence of 1.0 to an instance, when a human instruction of the respective modus was truly given, a confidence of 0.0, when no instruction was truly given. However, we expect models to not assign perfect probabilities and therefore not deliver a perfect performance. Instead we assume an accuracy of predicting whether or not an instruction was given of 0.95 and an average confidence 0.95 in its decision. Analoguously, we do not expect previous teams to assign perfect Coordinate Confidences given a valid instruction. We expect the Language & Vision and the Gesture Team to have an accuracy of about 90 % and to have an average confidence of around 0.90 in the piece they predict.

Given these assumptions, a consistent multimodal human instruction regarding a puzzle piece can be interpreted differently and possibly leading to conflict. Ideally both models assign the highest Coordinate Confidence to the actually intended piece and assign a high confidence to the predicted piece. In a less favourable scenario one team accurately and confidently predicts a piece while the other team inaccurately, but less confidently predicts another one. An unfavourable scenario would arise when one team accurately predicts a puzzle piece with low confidence while the other group predicts a wrong puzzle piece with high confidence, etc. Assuming an accuracy of 0.9 for the teams predicting piece, the worse the scenario the less probable it supposed to arise.

6.3 Data Simulation

To simulate natural data we first randomly determined whether an instruction was given and if an instruction for the Language & Vision or Gesture Team was given, which puzzle piece was intended (expressed in puzzle piece ID). Perfect models would translate this to ideal Instruction Confidences of 1.0 given an instruction and 0.0 given no instruction, as well as a Coordinate Confidence of 1.0 for an intended piece and 0.0 for all other pieces.

To mimic more realistic confidence values for Instruction Confidences and, given a true instruction, Coordinate Confidences, we draw values from a normal distribution. This distribution has a mean of the ideal confidence and a certain standard deviation that reflects the noise we assume for each previous model. However, this causes two problems. First, values can become smaller than 0.0 and larger than 1.0, therefore not reflecting probabilities anymore. Secondly, the values do not add up to 1.0 anymore, therefore also not reflecting the property of probabilities.

We addressed the problem by applying a softmax function to all Coordinate Confidence values. To apply this problem also to Instruction Confidences we needed to create an auxiliary value. We drew two values for each instance and each group: one value from a normal distribution with the mean of the ideal confidence that an instruction was given (Instruction Confidence) and second value from a normal distribution with the mean of the ideal confidence that no instruction was given (auxiliary value). The softmax function was applied to both values. Subsequently, we only used the first value reflecting Coordinate Confidence.

Softmaxing over values that were ranging around 1.0 and 0.0 caused the softmaxed values to be too similar to one another, not reflecting the property of DL or RL models of being rather confident in whatever they predict. To mimic this "self-confidence" of DL and RL models, we used a "scale-up-factor", with which we first multiplied ideal confidence values, that were then used as a mean for the mentioned normal distributions.

When no human instruction of a relevant modus was given, we generated Coordinate Confidence values by sampling from a uniform distribution and softmaxing the sampled values.

To align with our assumption regarding the accuracy and confidence of previous models and assuming a total number of 15 puzzle pieces, we tuned the following further hyperparameters:

Table 3: Hyperparameters for Synthetic Data Generation

	Standard Deviation	Scale Up Factor
Instruction Confidence	4	3
Coordinate Confidence	0.075	13

Based on this method of simulating confidence values we created 2 Mio. instances to provide sufficient data to train a DRS with RL. We stored the confidence values along with the ideal confidence values that reflect the true (yet simulated) human instructions.

6.4 Future Work

The current generation of synthetic data is based on multiple (over-)simplifications and needs to be improved in general and to develop more sophisticated DRS approaches.

Discontinuous Input. Since ASR-dependent groups do not continuously output, data should also reflect missing values i.e. with zeroes instead of always assuming extremely noisy values that are based on a uniform distribution.

Uninformed Predictions. In case previous groups assign a high Instruction Confidence without a true human instruction of the respective instruction modus, noisy output should also be drawn from a normal distribution, however with a much higher standard deviation. This way the property of DL and RL models of being rather confident even in wrong predictions could be imitated.

Gesture Team's Output. The Gesture Team is predicting one puzzle piece directly instead of first assigning Coordinate Confidences to each piece. To fit the desired output structure, a Coordinate Confidence of 1.0 was assigned to the predicted puzzle piece and a Coordinate Confidence of 0.0 to all other puzzle pieces. This should also be reflected in synthetic data.

Hyperparameter Tuning. The current dataset is based on our naive assumptions about accuracy and average confidence of previous team's models. These assumed values should be replaced with empirically estimated values as soon as previous groups have been fully integrated and hyperparameters should be tuned accordingly.

Progress in the Pentomino Game. Currently all training samples reflect the scenario of picking up the first piece when playing the Pentomino game. Proceeding in the Pentomino game, the number of available puzzle pieces decreases and therefore the probability to predict the correct piece as well as the Coordinate Confidence for the predicted puzzle piece should increase throughout the game.

Actual Human-Robot-Interaction. Up to this point we simplified HRI to the human presenting an instruction and the robot trying following it. Ideally dialogue would be extended to more than one round. I.e. an instruction would be presented and when the robot communicates uncertainty the human could express the same instruction again, resulting in the DRS receiving new confidence values, however based on the same original true instruction. This should be reflected in synthetic data and could be implemented in RL when determining state transitions in the environment.

7 Baseline: Rule-Based Decision Tree (JKT)

The rule-based method while not the most impressive method was chosen to be the baseline for the simple reason that it is the quickest to implement and prototype. Since the overall complexity of our project was within the ability of humans to distinguish out rules depending on the input, it was a fairly easy choice to make to first start off with a rule-based Dialog Manager as the baseline.

Another key aspect that the rule-based system allowed us to move forward with our plans initially was simply the rules being data agnostic, which was an important aspect since both the other methods,

Deep Learning and Reinforcement Learning are trained on synthetic data. Thus this baseline was meant to serve as a baseline for not only the two more complex methods but also as a measuring stick to analyse if the synthetic data prepared for this project reflect the ideas we initially had entering this project.

7.1 Literature Survey

In recent times, the usage of hand-crafted rule-based systems is quickly dwindling due to the extremely taxing nature of actually creating rules that can fit a system, but rule-based systems are still fairly common in cases that do not have a lot of complexity. A complex use case in general is made simpler by decreasing the level of information that is either required by the Dialog Manager or the information that is output from the Dialog Manager as seen in *Voyager* (Glass et al., 1995).

Within the realm of looking at Dialog Managers that work with hand-crafted rules, we can split on a further distinction of whether the Dialog Manager uses some form of state tracking. The simplest way to represent these rules is to organize them as pattern/response pairs that take the user's speech and provide the associated response, performing NLU, DM, and NLG tasks simultaneously. This does however limit the overall scope of the system since there are no states and as such, all the cases must be thought of beforehand to avoid failure. The best examples of such as system would be ELIZA (Weizenbaum, 1966) and ALICE (Wallace, 2003), one of the first conversational systems that were produced.

While this form is limited, it does offer some advantages that can not be found in other more complicated methods. Since the rules are handcrafted based on single utterances, it is excellent for quick bootstrapping by matching patterns in the utterances because it is simple to apply and does not need any training data. Additionally, it gives the people building the system a simple approach to incorporate subject expertise, and thus creating a knowledge base becomes a trivial task.

The next form of the rule-based system differs only in the way that it does include state tracking, and that allows the Dialog Manager to gain information from more than just the last utterance. By passing a sequence of utterances to the Dialog Manager, it is possible to build a finite state machine that is capable of following through the entire conversation, pattern matching each utterance to transition to the next state. A scenario where this is commonly implemented is ticket systems where the range of different utterances is quite limited and it is possible to guess what the user might say next in regards to previous utterances (John et al., 2017; Fast et al., 2018).

While this method of creating a rule-based Dialog Manager has all the same advantages of building a Dialog Manager as the previous method, it does include its own issues. The major shortcoming that can be seen in this method, one that is synonymous with all finite state machines, is the system breaks when a transition between states is not considered. By virtue of how finite state machines are created, they are quite linear in manner, and if the user of the Dialog Manager answers in a way that does not have any transition from the current state, the system will not be able to bypass it (McTear, 2002).

More recently, some work has also been done on integrating Pre-trained Language Models with rule-based systems to allow for more elaborate use cases that cannot directly be tackled by either of the two methods mentioned above. Due to its interpretability, rule-based conversation management remains the most often used approach for commercial task-oriented dialogue systems. In contrast, data-driven dialogue systems—which typically have end-to-end structures—are more commonly used in academic research and are better equipped to handle complex discussions. However, these techniques need a lot of training data and the behaviours they produce are less interpretable.

The Carina Dialog System build by Quan et al. (2021) mitigates this issue by combining BERT with more traditional rule-based Dialog Managers. In the realm of our project, however, this method is not applicable since we do not have end to end data that will be required to train such as system.

7.2 Proposed Model

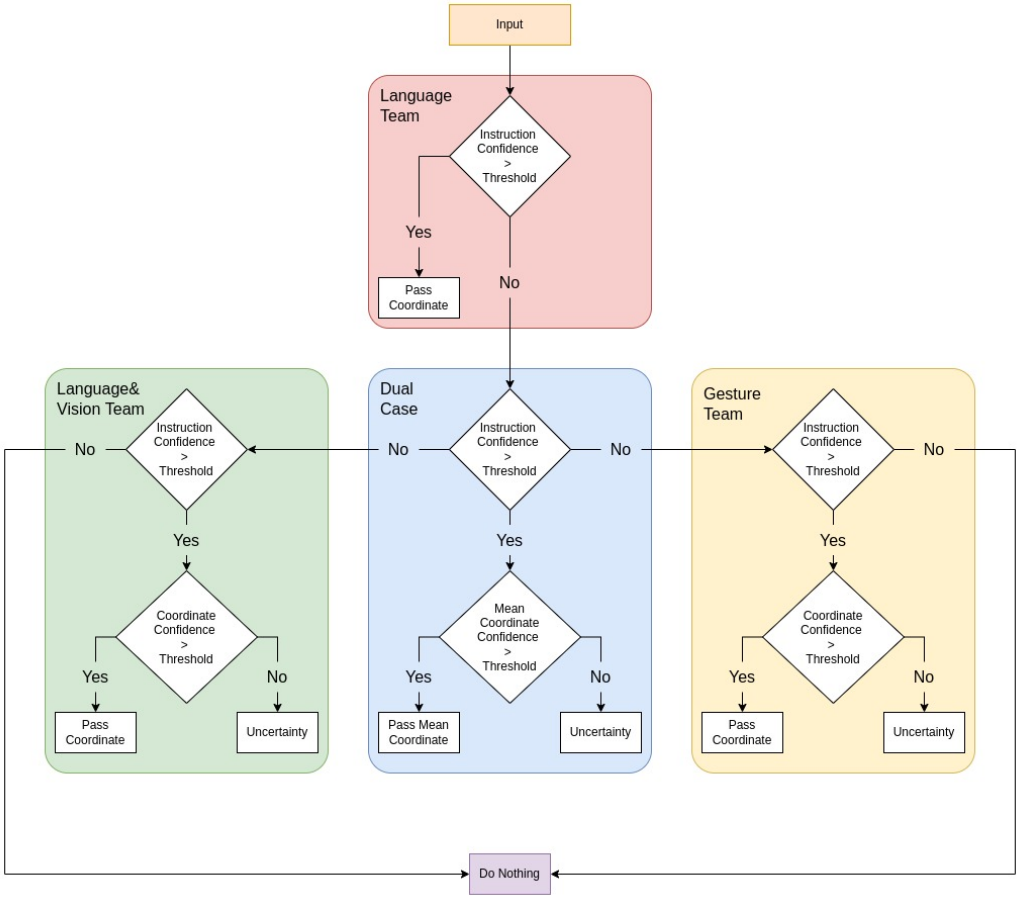

Figure 7: Decision Tree

The first approach that we decided to try out was to use a decision tree, Figure 7, to try and reach a decision on which of the three teams to listen to. As seen in Section 6, our Dialog Manager gets input from three discrete teams that run independently of each other on almost mutually exclusive data.

Another major factor in making the decision tree was the fact that we wanted to make a baseline that could be compared to more complicated but arguably better methods such as Deep Learning methods or Reinforcement Learning methods which will be discussed in the following sections.

The decision tree itself has two manually set thresholds, the instruction confidence threshold and the coordinate confidence threshold. The former is used to check if any team is confident enough in their input to the DM, and if so, the input is actually taken into consideration. The latter performs a similar check, this time, however, on the confidence of the pieces themselves. This method allows the decision tree to be tuned to a fairly decent amount and allows for much more granular control over what is being passed on to the Motion Team. In all cases where an input passes the instruction confidence check and yet the coordinate confidence is lower than the coordinate confidence threshold, the dialogue manager produces an *UNCERTAINTY* output.

Since we were working with three different teams, one of our first goals was to find out if there was any order of priority in terms of the input received between the teams. As you can see from Figure 7, in our project we have chosen the Language Team to get first priority to get processed and use their information to pass on to the following teams. The reasoning behind this choice is quite straightforward as the Language Team is solely responsible for keywords such as *STOP* and *GRAB* which regardless of any other team's input must be listened to.

Another case where a lot of consideration need to be put in was in handling the input when two or more teams were confident in their instruction. In our case, the only two teams that could have input with high confidence occurring together were the Language & Vision Team and the Gesture Team as the decision tree handles the Language Team separately.

While this particular scenario had the most amount of reworks and testing with data, we ultimately settled on a simple yet effective method of averaging the per piece confidences and checking if that was above the coordinate confidence threshold that we had set.

Moving on, the next two cases are for the Language & Vision Team and the Gesture Team respectively. Both of these cases are only processed if and only if the Language Team does not have any information with high confidence or does not pass any new information at the current time. Similar to what was in the case of the Language Team, we check to see if the confidence of the instruction received from either team is higher than our threshold and if so, the decision tree uses the coordinates given by the team.

And finally, in the case where no team has instruction confidence that is higher than the instruction confidence threshold the decision tree does not pass any action on to the Motion Team.

7.3 Results

The decision tree while being fairly simple by design, mostly due to the virtue of having only three input channels and limited action space, performs quite well. While a lot of manual fine-tuning of the instruction confidence threshold and the coordinate confidence threshold, the decision tree performed

beyond expectation on the generated synthetic dataset, Table 4. Although, the hyper-parameters used to tune the decision tree will face the same issue as the methods in the following sections, and that is the fact that these hyper-parameters are dependent on the dataset itself.

Table 4: Decision Tree Hyper-parameters and Results

Instruction Confidence Threshold	Coordinate Confidence Threshold	Accuracy
0.8	0.7	94.50%

7.4 Discussion

One aspect of the rule-based decision tree that makes it an appropriate approach amongst the different methods we tried is the fact that it produces inferences extremely quickly. On average the decision tree takes no measurable amount of time while make a decision. This makes it uniquely suited to an incremental system as any overhead in processing time will slow down the entire system. By being able to process immediately, we can ensure that the system can react as soon as possible making the robot seem more natural.

That said, the reason the decision tree is so fast is that the tree itself is fairly simple. The tree is based on hand-crafted rules limiting this method severely allowing for no change to occur in the inputs. Any change in the number of input teams or if the whole system is expanded to be a multi-round system, rather than a single-round system as it is now, will cause this method to completely fail.

8 Dialogue Policy Learning (SH & CW)

8.1 Our common Process of Policy Learning (SH)

In this subsection, we will describe the common process of Policy Learning for both Deep Learning and Reinforcement Learning. As for the Policy Learning in DM described in Section 2, Deep Learning and Reinforcement Learning are mainstream training methods (Ni et al., 2022). For each method, we are using the same synthetic data, which will be explained in Section 6. As we described in Section 2, we implemented two tasks to make the policies with each training method.

- Task 1: Predict Action
- Task 2: Predict Uncertainty of the Action

Action will be described either by the numerical ID of the pentomino piece that the robot arm picks up, the action ID that follows the Language Team's relative vector movement, or no action ID. Uncertainty will be expressed by the number either 0 or 1, reflecting how confident the models perform on task 1 given the multimodal information.

At the beginning of our Policy Learning generating process, we only prepared task 1, which outputted only the piece and action ID except for uncertainty since we had not noticed that implementing only task 1 cannot give the uncertainty flag to the Motion Team. However, in each method of Deep

Learning and Reinforcement Learning, we need to implement two models for each task since we need to predict with the new model for task 2 without reusing model 1.

There are two reasons behind this idea. Firstly, we cannot combine the uncertainty model with the action model because we need to predict the uncertainty with the model that has not seen the actual action (y label). However, the first model which was trained to predict the action has seen the synthetically generated y label. To predict uncertainty in the real user's environment in the future, DM has to output the uncertainty based on only the input (X features) of DM without seeing the y label.

Secondly, as you can see in Section 6, our synthetic dataset does not include an uncertainty label. Thus, uncertainty should be decided based on the first model's correlation between y prediction and y label given each instance's X features using the new model which has not seen the dataset yet when they predict the uncertainty.

We processed the six steps below for implementing both tasks in Deep Learning model and Reinforcement Learning model. The visualised image of the step 4, 5 and 6 is in Figure 8.

1. Separate the dataset into two parts, so the first one (dataset 1 with X1 and y1) is for task 1, and the other rest (dataset 2 with X2 and y2) is for task 2.

2. Split each dataset into training data and testing data.

3. Conduct task 1 on dataset 1 and get a prediction on action (y1_pred) and model 1.

4. Conduct task 1 again on dataset 2 using model 1 and get a prediction on action (y2_pred).

5. Generate the additional uncertainty label in dataset 2 based on the comparison between the y2 and y2_pred. For the uncertainty label, it is 0 when y2_pred is the same as y2 (not uncertainty), and 1 otherwise (uncertainty). Uncertainty label becomes the new y label in task 2. (Uncert_label)

6. Train with dataset 2 with X2 and the Uncert_label and get an uncertainty label prediction (Uncert_label_pred) finally.

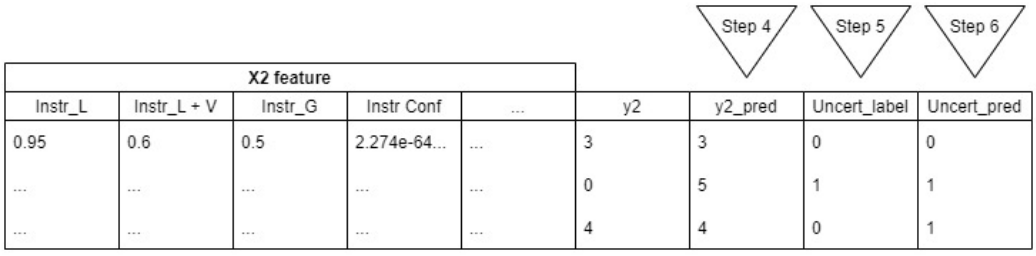

Figure 8: The image of dataset 2 for the step 4, 5 and 6

8.2 Deep Learning Method (SH)

8.2.1 Related Work and Motivation on Deep Learning Policy Learning

There are mainly two factors that motivated us to use Deep Learning Method for dialogue Policy Learning. The first factor is the Deep Learning performance in general and the availability of our dataset. According to Ni et al. (2022), Deep Learning policy models perform well; however, the training process totally depends on the quality of training data and the annotated datasets, which require intensive human labor. For our team, this was a concerning point of the Deep Learning method, since there is no natural data, and then we were not sure about what kind of training dataset should be prepared for Policy Learning. However, since we could make enough amount of the synthetic dataset with the automated system trying to reflect the real-world noise as much as we can, as in Section 6, we chose Deep Learning with a simple structure as one of the learning methods.

The second factor is the need to compare the simpler model with the Reinforcement Learning mechanism considering the dataset structure and its generating process. Since recent Policy Learning studies show the highest prevalence of Reinforcement Learning as a learning method (Ni et al., 2022), our first approach was only focused on Reinforcement Learning in the beginning. However, our dataset instances are generated based on the idea that each instance is independent and they are not sequential, so we started to doubt if Reinforcement Learning works with the dataset because of its mechanism, as in Section 8.3. Also, it is interesting to compare the elaborate learning method with a simpler structural type of method with less computational cost, since Williams and Zweig (2016) mentioned that Deep Learning and Reinforcement Learning are complementary on Policy Learning, and Deep Learning alone without Reinforcement Learning can derive a reasonable policy even from a small number of training datasets.

8.2.2 Our Approach using FFNN

We built a Deep Learning model with feed-forward neural networks (FFNN) with multiple hidden layers and activation functions. According to Aldakheel et al. (2021), FFNN is proven to be efficient for learning the complex input-output relationship, particularly when there is no history dependency in the dataset. Since our dataset instances are independent with non-linear relations between input and output, and do not store their time history, we assumed FFNN serves the reasonable performance together with less computational time, which we hypothesise to affect the entire DM running time for each inference.

Typically, there are alternative ways of using Recurrent Neural Networks (RNN) or Convolutional Neural Networks (CNN) instead of FFNN depending on the task (Ni et al., 2022); however, we did not choose them because of the dataset structure as we mentioned above. RNN performs well when the input data are sequential because the hidden layers contain the previous states' information and they output based on the previous sequential information. CNN performs well when the dataset contains positional and hierarchical features because of the sliding window system filtering the local

features and pooling layers, which capture specific inner structures of the dataset. Because of such a mechanism, neither of the models motivates us to choose them for our Policy Learning.

8.2.3 Hyperparameters for Deep Learning Model

Below, we put all the hyperparameters in Table 5.

Table 5: Hyperparameters for task 1 and task 2

	Output Size	Hidden Layers	Hidden Size	Batch Size	Learning Rate	Epochs
Task1	15 + 2	2	32	512	0.01	10
Task2	2	2	32	512	0.01	10

As for the output size, 15 + 2 in task 1 indicates the number of piece IDs 15, decided by Language & Vision Team or Gesture Team, and two labels "No instruction" when there is no instruction given by each group and "Language Team's Action" that indicates to follow Language Team's output. Output size 2 in task 2 indicates the uncertainty label as either 0 or 1. As for the hidden layers, Ni et al. (2022) mentioned that Deep Neural Network's "Deep" refers to the fact that they include multi-layers, and its performance is one of the most powerful models. Thus, we set the hidden layers size 2 to examine the shallowest structure of the Deep Learning model to give a contrast with the nature of the Reinforcement Learning model's complexity. Hidden sizes and batch sizes are tuned among 2^n where n = 5..12. Learning rate is also tuned with 10^{-n} where n = 1..6. Number of Epochs is decided because of computational time and our increased dataset size. We assumed that more epochs might enable the models to learn more; however, with the trade-off of the computational cost, the number is chosen. Also, we increased the dataset size, which increases the number of weight updates; however, we assume using different samples should result in better test accuracy rather than multiple times training with the same samples, which might cause the model to overfit the smaller amount of dataset .

8.2.4 Result and Discussion

In Table 6, the result of both task 1 and task 2 are shown. In both tasks, the performance is significantly high, with an accuracy of 97.49 % and 97.56 % for each task. As we described in the former section, Deep Learning model's performance is basically depending on the dataset; therefore, we were not sure about the performance whether it could be very high or very low because of our synthetic dataset. It is because, as we described in Section 6, the y label depends on the artificially generated noise by our team, which we were not sure how much it would reflect the real-world natural data. However, this result shows that the simple Deep Learning model performs already high scores with only two hidden layers trained on the synthetic dataset. From this result, we assume that the accuracy is still high in the natural data

Table 6: Accuracy for task 1 and task 2

Time Cost	Task	Accuracy
37.34 s	Task1	97.49 %
	Task2	97.56 %

when the artificially generated noise reflects the real-world noise. When the synthetic generated data is biased compared with the natural data, our performance might be extremely different from the performance.

Also, the computational time cost is noted in Table 6. Because of its simple structure, the computational time of the Deep Learning was only 37.34 seconds, including the time of loading the dataset, training both models and evaluations. As in Table 7, the average DM running time for each inference with Retico is shown. As expected, the rule-based system runs each inference the fastest on average with 0 seconds, and Deep Learning and Reinforcement Learning follow it with 0.7 milliseconds and 1.3 milliseconds. Even though we suppose that the tiny time gap between each model does not bring a huge difference to the entire DM system, we find that this result reflects what we hypothesise, where Deep Learning method runs relatively less time cost compared with Reinforcement Learning with a more complicated learning structure than Deep Learning. From this experiment, if there is an advantage of using Deep Learning in our task and dataset, it is the cost efficiency with reasonably high performance and less computational cost.

Table 7: DM Average Running Time

	Rule	DL	RL
Time (ms)	0.0	0.7	1.3

8.2.5 Future work using Deep Learning

There are two suggestions for our future work. Firstly, we can try to build RNN based Deep Learning model. It is because our future dataset will be sequential, where the instances are sequentially dependent on enabling multi-round DM. Thus, along with the future dataset, our future work in Deep Learning should use RNN model because of its structure which enables output from the sequential hidden layers of information, containing the previous hidden states information.

Secondly, as we described in Section 2, we need to output the Coordinate Confidence, which indicates the confidence of the action prediction with a number ranging from 0 to 1. However, at the moment, we are always outputting 1 as Coordinate Confidence when we are "not uncertain" because of the uncertain label prediction system. To give the decimal number in our Deep Learning model, we can simply try to make a regression model instead of a classifier model without taking the $argmax$ function on the Uncertainty label. However, since we cannot compare the result with other types of learning methods in that way because of their implementation mechanism, we will put this as one of our future works.

8.3 Reinforcement Learning (CW)

Reinforcement Learning (RL) is a family of machine learning approaches to control dynamic systems (Scheffer, 2019). Deep RL is a family of approaches that combine RL with DL and are responsible for the enormous success and popularity of RL-based approaches in recent years(Ni et al., 2022). Today RL approaches are used to control robots in simulations, play sophisticated strategy games such as GO or Shogi and guide robots interaction with humans (Achiam, 2018). Most of the current

techniques to train DM, specifically DRS, are based on RL, which is why we chose RL as one approach to train our DRS.

Since we needed to learn the concepts of RL, its algorithms and implementation from scratch, we will first present RL's key concepts and algorithms. Subsequently, we will propose current RL techniques for DM. Based on this we will reason how we framed HRI for the Pentomino game as an RL game and how we chose to train our DRS. The general approaches we took as well as results will be discussed and finally we will suggest further steps to improve our RL model.

8.3.1 Key Concepts

In its essence an RL-based model can be conceptualized as an **agent** that "lives" in a world, a so-called environment. This environment can change based on the agent's **action** and/or on its own accord — like in real life. In our DS the DRS can be conceptualized as agent that "lives" or operates in the environment that is constituted by the fellow human player. (Sutton and Barto, 2018)

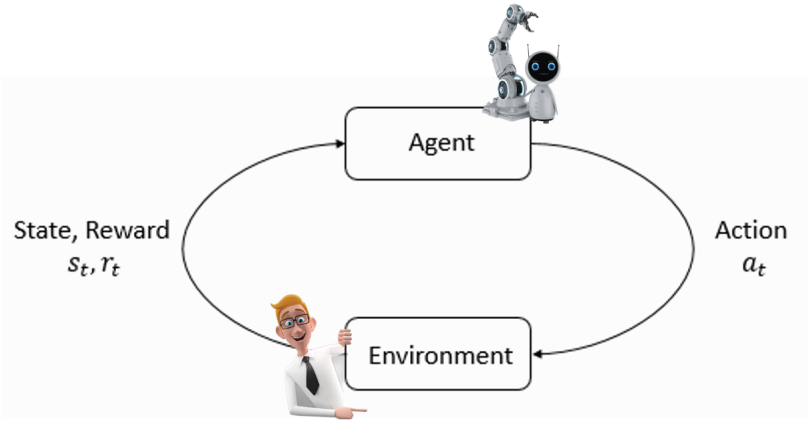

Figure 9: Flow of Information in an RL-game.

In general different states of an environment can be differently rewarding to the agent. The agent can either **fully** or **partially observe** how the world around it changes and how rewarding or unrewarding a certain state of the world is. The agent's goal is to maximize its accumulated and into the present discounted **reward**, which is called **return** (Sutton and Barto, 2018).

Via trial and error learning the agent tries to find rules. These so-called **policies** are learned to guide the agent's future decisions. In deep RL these policies depend on a set of parameters θ that can be learned with DL networks. **Value functions** describe the value of a state. This value is defined as the expected return when being in the state of interest and afterward forever following a certain policy. The so-called Optimal Q-Function describes the return the agent earns when starting in an arbitrary state, choosing an arbitrary action and afterwards following the optimal policy given the environment. These concepts are used to ultimately optimize the agent's behaviour to maximize its return (Sutton and Barto, 2018).

8.3.2 RL Algorithms

To identify suitable algorithms for our DRS, we used the Taxonomy of RL algorithms, provided by Achiam (2018). Based on recent work that will be described in section 8.3.3 and our acquired understanding of RL-algorithms we made three decisions between RL-algorithms:

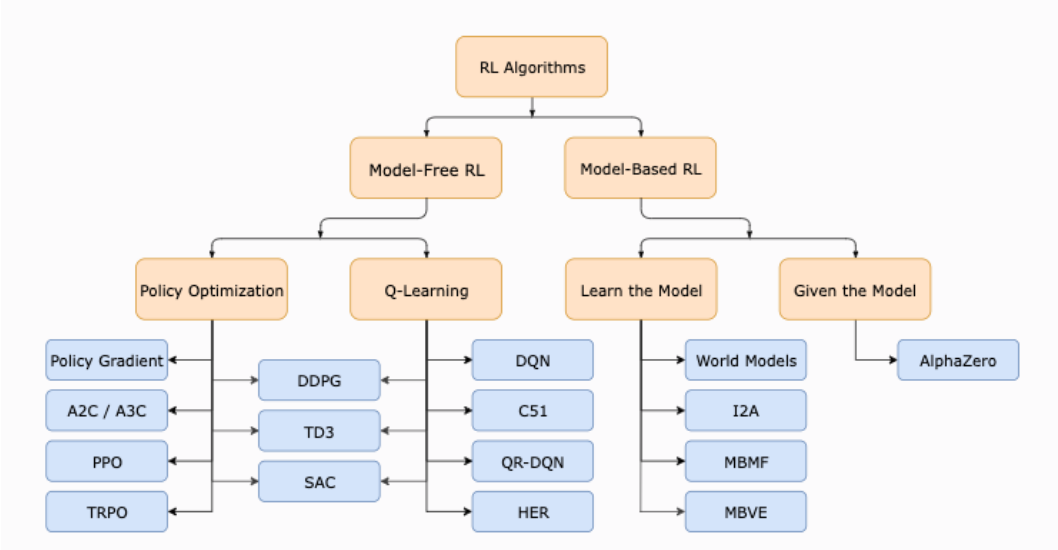

Figure 10: Taxonomy of RL Algorithms (Achiam, 2018)

Model-based vs. Model-Free RL. In model-based RL a model of the environment is learned, which enables the agent to plan. The model is constituted by the transition probability distribution and the reward function. Model-free RL manages to infer without such a model. Model-based RL enables agents to explicitly weigh available actions and their implications. This allows an agent to plan ahead and leads to sample efficiency during training. It was successfully used in AlphaZero, a program that mastered e.g. Go or Shogi. However, in many application cases it seems to overfit to the point, where models completely fail in a real environment. While being less sample-efficient model-free RL is easier to implement and less prone to overfitting. (Achiam, 2018; Huang, 2020)

Policy Optimization vs. Q-Learning. Model-free RL is further divided into the families of Policy Optimization and Q-Learning. With Q-Learning an Optimal Q-function is estimated and optimized. While being more sample-efficient, performance stability is dependent on how well the Optimal Q-function can be estimated. Policy Optimization on the other hand optimizes the agent performance directly, resulting in more stable and reliable performance. For Policy Optimization a policy is explicitly represented and optimized to maximize return (Achiam, 2018).

A2C vs. PPO. Stablebasline3 provides implementations of the algorithms A2C and PPO. The A2C algorithm maximizes the return w.r.t. the policy. The PPO algorithm is maximizing a surrogate objective function. This surrogate function is not directly measuring performance, but conservatively estimates the change in performance with each update. According to Tewari (2020) the PPO algorithm provides a better performance rate than A2C (Achiam, 2018; Tewari, 2020).

8.3.3 Recent RL Techniques for DM

Recent publications about DM training mostly report RL as training method. Ni et al. (2022) reason that the structure of a DM perfectly aligns with the characteristics of RL: an agent observes a current state of the environment (comparison: DST) and needs to choose an action given that observed state (comparison: DRS). Different techniques have been developed to further improve RL-based DM training (Ni et al., 2022).

To train an agent to optimally behave in a certain environment, it needs to be determined how the environment is behaving. Because large naturally generated data is often not attainable, several approaches have emerged to simulate environment behaviour. Rule-based simulations can require an unattainable amount of expertise (Takanobu et al., 2020). To avoid the necessity to first create an environment simulator before training an agent, Takanobu et al. (2020) and Papangelis et al. (2019) propose Multi-Agent Dialog Policy Learning (MADPL). With this technique RL is extended to an N-agent setting in which agents learn their respective policies simultaneously. Papangelis et al. (2019) propose Q-Learning to solve this Multi-Agent Problem. To make the model scalable to complex tasks such as multi-domain dialog Takanobu et al. (2020) chose an actor-critic framework over Q-Learning.

Huang et al. (2020)'s RL approach adressed the problem of sample inefficiency when receiving feedback only at the end of a multi-domain dialogue. Huang et al. (2020) proposed reward learning to provide turn-by-turn rewards. To learn rewards one could annotate state-action pairs of human-to-human dialogues. However, this could be unfeasibly labour intensive. This problem could be mitigated with a novel approach for semisupervised policy learning, called Act-VRNN. With Act-VRNN a reward function is learned that models dialogue progress via expert demonstrations, either with or without annotations. With this method reward was estimated effectively and stably (Huang et al., 2020).

Wang et al. (2020) considered the phenomenon that in human-human interaction one human often predicts the other human's next action and already prepares for its own action based on that. This notion was incorporated in the deep RL method MCTS-DDU that combines an advanced form of Q-Learning, Double-Q Dueling Network (DDQN), with a Monte Carlo Tree Search (MCTS) algorithm. First the DDQN was optimized via direct RL while no planning was executed. For testing, the agent performed MCTS which allowed for decision time planning. In decision time planning an agent searches for the best action for any state, thus choosing an action more "mindfully" (Wang et al., 2020).

There are far more current techniques. Another often adressed problem is domain adaptability, that can be mitigated with approaches such as training the DST and DRS jointly (Le et al. (2020b) or incorporating knowledge graphs. Since our DS does not require domain adaptability and since our main goal is to understand the fundamentals of RL and develop our DRS step by step, we will not present further sophisticated techniques but dive into our own implementation.

8.3.4 Implementation

We created our RL models with Stable Baselines3 (SB3), a set of reliable implementations of RL algorithms in PyTorch, as well as the ‘Env‘ environment class from OpenAI Gym. We created several “generations” of an RL-based DRS:

1. **Single-Round & Single Input**. First, we conceptualized an overly simplified version of playing the Pentomino game to familiarize ourselves with SB3 and Custom Environments. We simplified dialogue and created a very simple RL game: the human gives one instruction to pick up a certain puzzle piece and the robot tries to follow it. When the robot chose an action the game ends — either successfully or not. Since only one turn was taken in this dialogue this is referred to as single-round game.
2. **Multiple Input (MVP)**. We elaborated on that exercise and created a MVP that is still treating HRI as single-round games, but processes all relevant output of previous groups.
3. **Uncertainty**. The single-round model was then complemented with another model to predict uncertainty of an action prediction.
4. **Multi-Round**. Finally, we conceptualized how to further develop a simple multi-round game that an agent can be trained on in the future, described in section 8.3.6.

Single-Round & Single Input

We solved the above described simplified problem by first subclassing the ‘Env‘ environment class from OpenAI Gym and defining our own environment, then choosing a suitable RL algorithm and training and evaluating our agent.

Environment Customization. In the environment we first initialized and defined the action space, observation space and a start state. The action space was of the type ‘Discrete‘, a list of possible actions, where in each timestep only one of the actions can be used. We allowed for as many actions as there are puzzle pieces, expressed with integers from 0 to n -1 puzzle pieces. For the observation space we chose ‘Box‘, an N-dimensional box that contains every point in the action space. This space reflects the environment the agent can observe to base its decision on, namely the state. The start state was intitialized and later defined in the ‘reset‘ function, a function that resets the current state to a start state. This start state was defined as a vector of Coordinate Confidences, which we randomly generated within the environment in every time the reset function was called. The actual learning took place in the ‘step‘ function: here we compared an action the agent sampled from the action space with what we determined to be the correct action. In this simplified HRI scenario we determined the true action to be picking up the puzzle piece with the highest Coordinate Confidence. We rewarded the agent with +1, if its action aligned with this assumption and punished it with -1, if it did not align.

Algorithm Choice. After designing the environment we needed to train an agent with a suitable algorithm. In the presented related work model-free approaches were mostly chosen over model-based ones. Since model-based approaches are also prone to overfitting and our synthetic data is

based on our currently still naive assumptions about the previous team's output, we decided for a model-free approach. Next, we needed to decide between Policy Optimization and Q-Learning. The success of plain Q-Learning is heavily dependent on how well the Q-function can be estimated which can cause unstable and unreliable performance. To avoid this issue and to not get lost in more sophisticated forms of Q-Learning that adress this issue we chose PPO. Finally we needed to chose between the algorithms A2C vs PPO, the two options for Policy Optimization that were implemented in SB3. Due to a reported performance advantage of PPO over A2C by Tewari (2020) we chose PPO.

Training. We chose a number of 100.000 timesteps, a number that we assumed to be large enough for our agent to learn but not too large to consume too much time, since our goal was to generally familiarize ourselves with SB3 and create a proof of concept rather than an MVP. As a policy network we tried out a simple Multilayer Perceptron (MLP) with default parameters of 2 layers of 64 nodes. We logged all the results for that particular model by setting the hyperparameter verbose to 1. The other hyperparameters were set to SB3's default values which are rudimentarily listed in Table 8. Instead of optimizing an agent we trained to get hands-on experience, our priority was to create a MVP as soon as possible.

Table 8: Hyperparameters for Training an Agent with PPO

learning rate	n steps	batch size	epochs	gamma	clip range
0.0003	2048	64	10	0.99	0.2

Evaluation. We used accuracy as evaluation metric, since we do not weigh false positives other than false negatives and also do not assume significant class imbalance in our synthetic data. Since this implementation was supposed to only serve as exercise, we did not save the script or the evaluation, but further developed an MVP.

Multiple Input (MVP)

After familiarizing ourselves with SB3 and Custom Environments, we further developed the action space, start state and reward function to retrain an agent that serves as MVP. We extended the action space with the actions "do nothing" and "listen to the Language Team". We changed the start state to reflect the output of all teams which we generated outside of the environment. Further description of data generation is given in Section 6. The reward was now determined by whether or not the model predicts the "truly" intended piece which is determined during data generation, also described in Section 6. After defining the environments we trained the agent analog to the previous agent, but with 666.666 timesteps to reach better performance. The training took 921 seconds. The agent reached an accuracy of 0.9724.

Uncertainty Prediction

Analog to the DL approach we trained a second agent on predicting whether the previously described model would choose the right action (certainty) or not (uncertainty). To distinguish from the

previously described model we will refer to this model as “Uncertainty model” and the previous one as “Action model”. To train the Uncertainty model the same data was used as for the DL approach, the agent was trained with the same algorithm and the same hyperparameters as for the RL-based Action Model. The training took 818 seconds. The agent reached an accuracy of 0.9720. Both models together constitute the RL-based DRS that was used in our Retico-based DS. First, the Uncertainty model determines whether a prediction of the Action model is likely to be correct or not. If it is likely to be correct, the previous team’s output will be forwarded to the Action model that predicts an action. The average inference time for this DRS over 100 instances was 1.5 ms in Retico.

8.3.5 Discussion

Reducing a game to only one round defies the purpose of RL, which is controlling a dynamic system. However it serves as a starting point to adapt the environment for a multi-round agent as described in section 8.3.6. The agents trained with RL still performed with an accuracy above 97 % and only 0.27 percent points worse than DL, despite the fact that a singe-round game is basically a supervised learning problem that DL approaches are more suitable for. However, it remains unknown how well all our models perform on natural data. Since HRI is ideally dynamic and since RL is the dominant method to train dynamic systems in general and specifical to train DRS, we expect future multi-round RL models to outperform our DL-based and rule-based approaches.

8.3.6 Future Work

In the future the following aspects should be improved with:

- multi-round learning
- experimenting with RL algorithms and hyperparameters
- online learning, possibly with BCS

We came up with an again simplified way of framing HRI for the Pentomino game, but this time as multi-round game. Up to this point we treated “uncertainty” like the other actions as the end of a game. Instead it could also be treated as the end of a round that triggers another round that starts with the human expressing the instruction again — either by repeating, rephrasing it or switching the modus such as in the following potential dialogue:

Human:	Pick up the red puzzle piece.
Robot:	*communicates uncertainty*
Human:	“This one:”, *points at red puzzle piece*
Robot:	*picks up red puzzle piece*

The agent should have access to the dialogue history, which is constituted not just by the last instruction, but also by previous instruction(s) in that dialogue. The agent should figure out when to base a decision only on the newly expressed input and when to also take the previously expressed

instruction(s) into account to increase performance over the single-round approach. We need to grant the agent access to the dialogue history and further adapt the Custom Environment in the following manner:

Start state. A start state should be generated on the fly within the environment in the same manner it is currently generated outside of the environment.

Reward. In addition to rewarding a correct action and punishing an incorrect one, there should be a mild punishment for communicating uncertainty. Punishing uncertainty to some degree incentivates the robot to not communicate "uncertainy" too often since it would make it an inefficient partner in the Pentomino game. However, punishing uncertainty less than for predicting a false action causes the robot to not predict an action if there is too little evidence to substantiate it. The concrete extent of the uncertainty punishment would need to be manually tuned or learned.

State transition. With our current method of data generation, we keep track of the real content of an instruction, e.g. picking up a certain piece, and the team an instruction is relevant to, e.g. the Language & Vision Team. If a certain puzzle piece was supposed to be picked up, we can keep the ideal Coordinate Confidences and randomly choose whether the next instruction is coming from the Language & Vision Team, the Gesture Team or both. If initially an incremental position description was instructed, the true Instruction Confidence of 1.0 remains for the Language Team. We then again draw instruction and Coordinate Confidences from normal distributions in the same manner as for the start state generation.

These improvement suggestions are the next steps we would take given our current understanding about RL for DM in our DS. As related work shows, there are much more sophisticated approach out there. We would gladly participate in a subsequent project to dive deeper into RL.

9 Discussion (JKT)

Our primary goal in trying out three different approaches was to determine what could be the most suitable approach while still being able to be expanded further down the line. As seen from Table 9, all three approaches are perfectly suitable for use in our Dialog Manager and switching one out for the other does not provide any great advantages or disadvantages over the other methods. That said there are a few caveats in each method that we will be looking at now in this section.

Table 9: Summary of Results

Method	**Accuracy**	**Total Inference Time**
Rule Based	94.5%	Approx 0ms
Deep Learning	Task1: 97.49% Task 2: 97.46%	0.7ms
Reinforcement Learning	Task 1: 97.24% Task 2: 97.20%	1.3ms

Looking at the baseline strategy, there is not much to explicitly state that has not been covered yet. Since the main goal of the rule-based Dialog Manager was to allow for quick prototyping, this method did not focus much on how easy it would be to expand the system and into consideration more complex aspects of being a Dialog Manager for this particular use case. As such it would be practically impossible to further expand on this method if it goes beyond what is comprehensible by humans, i.e. if the problem gets more complicated.

That said, when looking only at the project in its current state, the baseline method holds up pretty well in terms of performance when comparing it to the other methods. The fact that it takes no measurable time to take a decision is especially one aspect that while not intended, is something that makes this approach quite useful. Since the Dialog Manager is set up using Retico checks and then processes new inputs every 0.1 seconds, having any overhead in taking the actual decision is quite undesirable.

The other goal of the rule-based baseline was also to be able to get a measure of how good the synthetic data itself is. Since the rules were made painstakingly based on logical conclusions, for the data to be deemed usable, the rule-based baseline should be able to achieve a relatively high accuracy score. Since the baseline was able to achieve 94.5% accuracy, we determined that the data itself was good enough to be considered a fair replication of the real-world input when the project is actually deployed.

Moving on to the Deep Learning approach, this method has far more versatility than the rule-based approach simply due to the virtue of not having any need for human effort to learn the strategies while taking a decision. As mentioned in Section 8.2, the model architecture was kept fairly simple to ensure that the model remains as small as possible and thus can take decisions as quickly as possible.

While the Deep Learning approach is quite simple it still results in the best performance on the synthetic dataset among the three approaches that we tested. Since these results are based on synthetic data, these results are reliant on the fact that our rule-based system does in fact consider all the scenarios Thus basing the result of the Deep Learning approach on this data would be reflected when considering real-world use cases.

Beyond this, the Deep Learning approach is also quite a bit more versatile. Not only can the current architecture be expanded to give us the ability to train the model on more complex scenarios, but the whole architecture is also based on the simplest form of Deep Learning, Linear Layers, as it is now. This makes the "Deep Learning" approach a misnomer as the whole architecture can very well be changed to more complex methods if needs be.

One pitfall of using Linear Layers to build the architecture is the fact that Linear layers are generally incapable of being able to learn on sequential data. Since the scope of the project as it is now only deals with a single turn of the game, using Linear Layers and not being able to handle more turns, i.e. sequential data, was a drawback we were willing to take to get better computational performance. If the scope of the project is expanded to include multi-turn games, we expect the results to be similar

if we use a recurrent model such as LSTM instead of the Linear Layers making the Deep Learning approach quite robust and future-proof.

Finally, the Reinforcement Learning approach is conceptually by far the most suitable method among the three we tested. Since this method is generally the best method to be used when considering a game-like input environment, it was a fairly easy decision to try this method out for our particular project. Needless to say, this approach performs very well on our synthetic data proving the previous statement to be a fact.

While Reinforcement Learning itself is an umbrella concept, we had to take into consideration the most suitable approach in Reinforcement Learning we could apply for our use case. Beyond what has been discussed in Section 8.3 PPO also has an advantage over normal policy gradient methods since they have an issue of exploding in computational time and space when considering large-scale problems, thus we chose to use PPO to get the best of both worlds.

PPO formalizes the limitation as a penalty in the objective function rather than imposing a strict constraint. We can utilize a first-order optimizer, such as the Gradient Descent method, to optimize the goal by not avoiding the constraint at any cost. Even if we might occasionally breach the condition, the harm is much less and the computation is much simpler.

We assume that the slightly lower score of the Reinforcement Learning approach is due to the nature of PPO occasionally breaching the conditions. However, as mentioned in Section 8.3, we can quite confidently assume that using PPO will be more beneficial when expanding the scope of the project, more so than the Deep Learning approach.

With all that said about PPO, one flaw that this approach has is the fact that inference times are slightly higher than the other two approaches. As of right now, in the scope of the project, the inference times are not to the extent where we sacrifice too much smoothness at the cost of performance and future expandability, thus we believe it to be the most logical approach for our project.

10 Integration (SW)

While we planned to eventually integrate all other teams with the Retico framework, this did not work out in all cases. To not be depended on the other teams' success, we did add our dataset to the Retico environment to be able to load random inputs from those teams that we could not integrate in time.

10.1 Language & Vision

Unfortunately integrating the Language & Vision Team was not possible so far but we are looking forward to integrating them once their code is made accessible to us.

10.2 Language

The Language Team made their code available quite early and we were able to successfully integrate them in our environment. We agreed with them to pass the transcribed text together with the `final` flag from the speech recognition IU into their provided function, which returns their interpreted vector with a confidence in their result. Because they decided to encode the special flags as vectors where all three values are identical, we added a translation into our flag format. Another small thing we had to change was that their model seemed to assign high confidences when their output was actually expressing no instruction given. We therefore filtered those cases and set the confidence to zero to make it work with our decision models. It does seem their function is only outputting something when the `final` flag is set to True though.

10.3 Gesture

We managed to integrate the Gesture Team shortly before the end of the project[2]. Since a full test would require the RealSense camera, the integration was only tested in demo mode, which uses a static image as the top-down camera and random pointing gestures for the intersection calculation. However, this should not affect the integration interface itself, so we expect it to work in normal mode as well.

Their function takes no inputs from our side, but required setting up the mapping and gesture recognition in the `setup` function of the Gesture Module. We use the generated map of pieces to build a bijective mapping of piece ids to coordinates. This is needed as our models work only with the piece ids but the Motion Team at the very end requires the coordinate of the piece. We query their provided function periodically and receive back a list that is either empty or contains one single coordinate. This differs from our initially assumed format of a confidence mapping over all pieces, so we translate this single piece into a one-hot representation for the Gesture IU. Their chosen approach also does not output an overall instruction confidence so this value is set to 1 when an intersection of gesture and piece was detected and 0 otherwise.

10.4 Motion

For the Motion Team, integration was foiled by issues with the ROS system. We don't fully understand where those issues lied but were told that interaction with ROS would not be possible within the project time, so we made no further attempts to solve this. As it stands, the Motion Module merely collects the passed actions in a queue and prints them to the console. We did however add the functionality of revoking an action, which removes the action from the queue.

[2] The Gesture Team did end up modifying their own Retico module in the final days of the project but this description is based on the integration we ourselves performed which can be found in the `group-B-dialog-manager` branch.

11 Conclusion (SW)

Our task was developing the dialog management for the cooperative pentomino puzzle game. To this end we used Retico, a modular framework for incremental processing, to build a pipeline of processing steps that represent the backbone of the project. This includes the handling of the automatic speech recognition system, forwarding its output to the Language and Language & Vision Teams and then taking their respective outputs, combined with that of the Gesture Team to make a decision on which action to perform. This decision is then passed on to the Motion Team.

We explored three options of decision-making models: A hand crafted rule-based decision tree, a Deep Learning model and a Reinforcement Learning model. While the decision tree was simple to build, the Deep Learning and Reinforcement Learning approaches required data to train. Since no real-world data was available, we came up with a way of generating a synthetic dataset based on what we expected other teams' inputs to look like. All three models perform at above 90 percent accuracy on the dataset, with the decision tree performing slightly worse than the other two models. There is however no clear winner as the downside of the trained models is their slightly higher inference times and their reliance on realistic data. The decision tree on the other hand does suffer from being much harder to adapt should real-world data become available or should the system be extended to process multi-round inputs.

References

Joshua Achiam. 2018. Spinning Up in Deep Reinforcement Learning.

Fadi Aldakheel, Ramish Satari, and Peter Wriggers. 2021. Feed-forward neural networks for failure mechanics problems. *Applied Sciences*, 11(14).

Jennifer E Arnold, Carla L Hudson Kam, and Michael K Tanenhaus. 2007. If you say thee uh you are describing something hard: the on-line attribution of disfluency during reference comprehension. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 33(5):914.

Julia Berg and Shuang Lu. 2020. Review of interfaces for industrial human-robot interaction. *Current Robotics Reports*, 1(2):27–34.

David DeVault, Kenji Sagae, and David Traum. 2009. Can I finish? learning when to respond to incremental interpretation results in interactive dialogue. In *Proceedings of the SIGDIAL 2009 Conference*, pages 11–20, London, UK. Association for Computational Linguistics.

Ethan Fast, Binbin Chen, Julia Mendelsohn, Jonathan Bassen, and Michael S. Bernstein. 2018. Iris: A conversational agent for complex tasks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, page 1–12, New York, NY, USA. Association for Computing Machinery.

James Glass, Giovanni Flammia, David Goodine, Michael Phillips, Joseph Polifroni, Shinsuke Sakai, Stephanie Seneff, and Victor Zue. 1995. Multilingual spoken-language understanding in the mit voyager system. *Speech Communication*, 17(1):1–18.

Julian Hough and Matthew Purver. 2014. Strongly incremental repair detection. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 78–89, Doha, Qatar. Association for Computational Linguistics.

Qingyan Huang. 2020. Model-based or model-free, a review of approaches in reinforcement learning. In *2020 International Conference on Computing and Data Science (CDS)*, pages 219–221.

Xinting Huang, Jianzhong Qi, Yu Sun, and Rui Zhang. 2020. Semi-supervised dialogue policy learning via stochastic reward estimation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 660–670, Online. Association for Computational Linguistics.

Rogers Jeffrey Leo John, Navneet Potti, and Jignesh M Patel. 2017. Ava: From data to insights through conversations. In *CIDR*.

Arne Köhn. 2018. Incremental natural language processing: Challenges, strategies, and evaluation. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2990–3003, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Bo Li, Tara N Sainath, Arun Narayanan, Joe Caroselli, Michiel Bacchiani, Ananya Misra, Izhak Shafran, Hasim Sak, Golan Pundak, Kean K Chin, et al. 2017. Acoustic modeling for google home. In *Interspeech*, pages 399–403.

Michael F. McTear. 2002. Spoken dialogue technology: Enabling the conversational user interface. *ACM Comput. Surv.*, 34(1):90–169.

Erinc Merdivan, Deepika Singh, Sten Hanke, and Andreas Holzinger. 2019. Dialogue systems for intelligent human computer interactions. *Electronic Notes in Theoretical Computer Science*, 343:57–71. The proceedings of AmI, the 2018 European Conference on Ambient Intelligence.

Thilo Michael. 2020. Retico: An incremental framework for spoken dialogue systems. In *Proceedings of the 21th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 49–52, 1st virtual meeting. Association for Computational Linguistics.

Thilo Michael and Sebastian Möller. 2019. Retico: An open-source framework for modeling real-time conversations in spoken dialogue systems. *Studientexte zur Sprachkommunikation: Elektronische Sprachsignalverarbeitung 2019*, pages 134–140.

Jinjie Ni, Tom Young, Vlad Pandelea, Fuzhao Xue, and Erik Cambria. 2022. Recent advances in deep learning based dialogue systems: A systematic survey. *Artificial Intelligence Review*, pages 1–101.

Maike Paetzel, Ramesh Manuvinakurike, and David DeVault. 2015. "so, which one is it?" the effect of alternative incremental architectures in a high-performance game-playing agent. In *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 77–86, Prague, Czech Republic. Association for Computational Linguistics.

Alexandros Papangelis, Yi-Chia Wang, Piero Molino, and Gökhan Tür. 2019. Collaborative multi-agent dialogue model training via reinforcement learning. In *SIGdial*.

Jun Quan, Meng Yang, Qiang Gan, Deyi Xiong, Yiming Liu, Yuchen Dong, Fangxin Ouyang, Jun Tian, Ruiling Deng, Yongzhi Li, et al. 2021. Integrating pre-trained model into rule-based dialogue management. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 16097–16099.

Kenji Sagae, Gwen Christian, David DeVault, and David Traum. 2009. Towards natural language understanding of partial speech recognition results in dialogue systems. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 53–56, Boulder, Colorado. Association for Computational Linguistics.

Tobias Scheffer. 2019. Online lecture on reinforcement learning.

David Schlangen, Timo Baumann, and Michaela Atterer. 2009. Incremental reference resolution: The task, metrics for evaluation, and a Bayesian filtering model that is sensitive to disfluencies. In *Proceedings of the SIGDIAL 2009 Conference*, pages 30–37, London, UK. Association for Computational Linguistics.

David Schlangen and Gabriel Skantze. 2009. A general, abstract model of incremental dialogue processing. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 710–718, Athens, Greece. Association for Computational Linguistics.

SC Stoness, J Allen, G Aist, and Swift Mary. 2005. Using real-world reference to improve spoken language understanding. In *AAAI Workshop on Spoken Language Understanding, Pittsburgh, Pennsylvania, July*, pages 38–45.

Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA.

Ryuichi Takanobu, Runze Liang, and Minlie Huang. 2020. Multi-agent task-oriented dialog policy learning with role-aware reward decomposition. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 625–638, Online. Association for Computational Linguistics.

Ujwal Tewari. Which reinforcement learning-rl algorithm to use where, when and in what scenario? [online]. 2020.

Richard Wallace. 2003. The elements of aiml style. *Alice AI Foundation*, 139.

Sihan Wang, Kaijie Zhou, Kunfeng Lai, and Jianping Shen. 2020. Task-completion dialogue policy learning via Monte Carlo tree search with dueling network. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3461–3471, Online. Association for Computational Linguistics.

Joseph Weizenbaum. 1966. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45.

Jason D Williams and Geoffrey Zweig. 2016. End-to-end lstm-based dialog control optimized with supervised and reinforcement learning. *arXiv preprint arXiv:1606.01269*.

Zhirui Zhang, Xiujun Li, Jianfeng Gao, and Enhong Chen. 2019. Budgeted policy learning for task-oriented dialogue systems. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3742–3751, Florence, Italy. Association for Computational Linguistics.