Addressing Ambiguity in Human Robot Interaction using Compositional Reinforcement Learning and User Preference

Jenalea Norma Rajab
562262

WITS
UNIVERSITY

Supervised by:
Prof. Benjamin Rosman
Dr. Steven James

Research Report submitted to the Faculty of Science, University of Witwatersrand,
in fulfilment of the requirements for the degree of Master of Science (Coursework and Research Report) in the field of Computer Science.

Abstract

The ability for social robots to integrate naturally with the lives of humans has many advantages in industry and assisted services. For effective Human Robot Interaction (HRI), social robots require communication abilities to understand an instruction from the user and perform tasks accordingly. Verbal communication is an intuitive natural interaction for non-expert users but it can also be a source of ambiguity, especially when there is also ambiguity in the environment (i.e. similar objects to be retrieved). Addressing ambiguity for task inference in HRI is an unsolved problem. Current approaches, that have been implemented in collaborative robots, include asking for clarifications from the user. Related research shows the promising results of using user preference in HRI, but no work has been found where user preference is employed specifically to address ambiguity in conjunction with clarifying questions. Additionally, these methods do not leverage knowledge learned from previous interactions with the environment and the life-long learning capabilities of Reinforcement Learning (RL) agents. Based on the related work and shortfalls, we propose a framework to address ambiguity in HRI (resulting from natural language instructions), that leverages the compositionality of learned object-attribute base-tasks in conjunction with user preference and clarifying questions for adaptive task inference. Evaluating our method in the BabyAI domain, we extensively test all components of our system and determine that our framework provides a viable solution for addressing the problem of ambiguity in HRI. We experimentally prove that our method improves user experience by decreasing the number of clarifying questions asked, while maintaining a high level of accuracy.

Declaration

I, Jenalea Norma Rajab (Student Number: 562262), hereby declare the contents of this dissertation to be my own work unless otherwise explicitly referenced. This research report is submitted for the degree of Master of Science (Coursework and Research Report) in Computer Science at the University of the Witwatersrand, Johannesburg. This work has not been submitted to any other university, nor for any other degree.

Signature

17 September 2024

Date

Acknowledgements

I would like to give a big thank you to Prof. Benjamin Rosman, Dr. Steven James and Geraud Nangue Tasse, for all their support and guidance and for always making themselves available to answer questions and supervise this research. To my husband and family, thank you for your endless support, questionable advice, and for pretending to understand what I have been working on. I could not have done it without your distractions and encouragement.

Contents

Chapter 1

Introduction

Robotics is an increasingly growing field, with applications in industry, medical services, collaboration and assisted care [Muthugala and Jayasekara 2018]. Due the inevitable prevalence of robots in daily life, Human Robot Interaction (HRI) is becoming a significant field of study. This field presents many research challenges, particularly concerning the requirement for robots to take instructions and perform tasks accordingly [Hatori *et al.* 2018]. Two types of HRI are broadly defined as non-verbal (i.e. gestures) and verbal, with verbal communication favoured as more intuitive natural interaction for users [Bamdale *et al.* 2019]. Ideally, we would want to seamlessly communicate with a robot, giving verbal instructions that do not require expert domain knowledge or unnatural specificity.

Giving an instruction to a robot, which references an object in the real world, requires grounded language learning, where the robot learns the meaning of the instruction by leveraging the context of its environment [Bamdale *et al.* 2019; Matuszek 2018]. Natural Language Processing (NLP) techniques can be used for such grounded language learning, translating from the natural human form into a simplified action command that the robot is able to perform [Shridhar *et al.* 2020]. These natural language instructions are called referring expressions, as they refer to an object the user would like the robot to act on and contain object descriptors that assist the robot in locating it within the environment [Doğan 2021, 2023]. Ambiguous instructions can form as a result of incomplete referring expressions, where only partial information is given, or where multiple objects in an environment match the same descriptor [Doğan 2021]. Incomplete referring expressions arise from faulty sensory inputs as well as the inherent ambiguity in natural language.

Current approaches addressing this problem of ambiguity predominantly include asking semantic clarifying questions (Q&A) [Doğan 2021]. The ability for the robot to ask a question helps with task efficiency and accuracy, rather than waiting for negative feedback after the task has been performed [Whitney *et al.* 2017]. These Q&A approaches make use of object detectors or defined object lists to determine the ambiguous object attributes and ask questions that consider the user perspective or gesture, to efficiently resolve the ambiguity [Doğan *et al.* 2022; Yang *et al.* 2022; Whitney *et al.* 2017].

Similar research in HRI around task efficiency and user experience show the benefit of using user preference in social robots [Mason and Lopes 2011; Lee *et al.* 2012a]. These systems keep track of a user's choices from previous interactions, utilising this knowledge to personalise the experience for the user [Lee *et al.* 2012a]. Such implementations show notable increases in user cooperation and engagement [Hellou *et al.* 2021]. However no work has been found where user preference is employed specifically to address ambiguity in conjunction with clarifying questions. The use of both clarifying questions and preference can help make robots more efficient and adaptable, in cases where the user is not in the same scene, and prevent repeated unnecessary clarifying interactions (i.e. if the robot knows that the user statistically prefers certain objects over others, it does not have to clarify the ambiguity every time this scenario occurs).

Additionally, it is noted that the current approaches studied to address ambiguity do not leverage the life-long learning capabilities of Reinforcement Learning agents to scale adequately to real-world non-stationary settings [Parisi; Akalin and Loutfi 2021]. Ideally, to resolve ambiguity, we would like the robot to not only learn user preferences and have the ability to ask clarification questions, but also leverage existing knowledge it has from interacting with its environment (i.e. object-based learned skills) to adaptively infer a task when ambiguity exists. In a real world environment it is infeasible to hard-code a robot for every possible task, user preference and question it may encounter or require. Reinforcement Learning (RL) is a common approach used to address this, in which a robot learns to take decisions or actions in order to maximise a numerical quantity. This approach however comes with the cost of long training times, large compute and generalisation issues [Cohen *et al.* 2021]. One emerging method, to aid in the goal of generally capable artificial intelligence, is compositional RL, which uses a logical combination of previously learned base-tasks to increase sample efficiency when a new task is presented [Tasse *et al.* 2020].

Consider an example domain shown in Figure 1.1, in which a non-expert user gives the robot an instruction to 'Pass the Cup', in an environment in which there are two possible options 'Glass and Porcelain'. The robot has learned the object-attribute base-tasks for 'picking up a cup', 'picking up a glass object' and 'picking up a porcelain object' from interacting with the environment, which can be logically combined using Boolean algebra to complete a task [Tasse *et al.* 2020]. In this case, where ambiguity exists, the system can determine from the learned base-tasks, that it needs to clarify whether to pick up the glass or porcelain object in order to infer the complete task instruction. The system will now consult the user preferences for the object materials accordingly and determine which one of the options has a higher preference probability. If there is uncertainty in the preferences (i.e. they are equivalent or within a certain range), a clarifying question can be asked (e.g. 'Would you like me to pick up the glass or porcelain one?'). The user preferences are adjusted accordingly following an interaction, to reduce the number of questions asked over time as the robot becomes more familiar with the user. Once the ambiguity has been clarified and the correct task has been inferred, the corresponding base-tasks can be composed to perform the task in the environment.

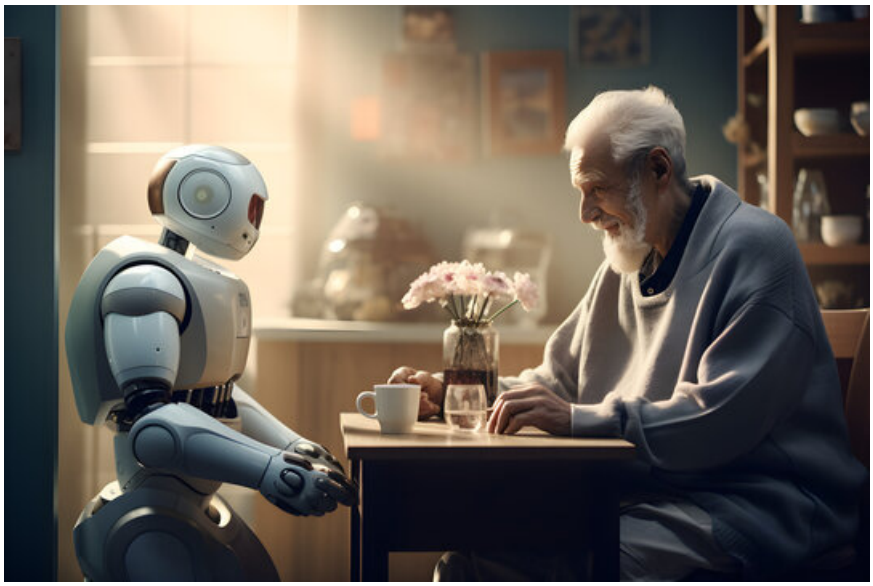


Figure 1.1: Robot interacting with an older man and assisting in completing task instructions, such as "Pass the Cup"

In this research we propose a framework that leverages the compositionality of learned object attribute base-tasks in conjunction with user preference and clarifying questions for adaptive task inference; to address ambiguity in HRI and perform tasks specified by a natural language instruction. More specifically, we use the approach defined in Cohen *et al.* [2021], to train specific object-attribute compositional value functions. This approach utilises a fine-tuned seq2seq transformer model to translate instructions from the user to tokens specifying the learned value functions [Cohen *et al.* 2021]. We extend this approach to allow for ambiguity in the referring expressions and leverage these learned base-task value functions to create relevant clarifying questions and track user preference to resolve ambiguity given the observable environment.

1.1 Contributions

We experimentally demonstrate that our proposed framework, leveraging learned compositional base-tasks in conjunction with user preference and Q&A systems, is a viable approach to address ambiguity in HRI. Our modular approach reduces the number of clarifying questions required and demonstrates task inference accuracy in line with unambiguous settings. This method also increases the ease of use for the user and provides a more personalised interaction over time.

Specifically our research makes the following contributions:

- We provide a practical application of computational RL in the field Human Robot Interaction.
- We leverage compositional RL to generate clarifying questions grounded in logic.
- We introduce the tracking of user preferences as a method to address ambiguity, and decrease the number of clarifying questions required over time.

In the next chapter the background and related work is provided for the main research areas. Chapter 3 defines research methodology for the various components of the proposed system. Once the research methodology is established, Chapter 4 provides the experimentation results. This is followed by the Conclusion in Chapter 5 and potential future research avenues.

Chapter 2

Background and Related Work

This chapter explains the background knowledge relevant to our methodology as well as the related work in the research area of addressing ambiguity in HRI. As outlined in Chapter 1, our approach utilises reinforcement learning to learn compositional task value functions. Natural language task instructions from the user are mapped to the relevant learned value functions by a fine-tuned transformer model. The agent then uses the compositional value functions to form policies and act in the environment to complete the task. Section 2.1.1 presents reinforcement learning and the sub-field of compositional reinforcement learning. This is followed by the required background information of transformer based language models in Section 2.1.2. An overview of related work in HRI and ambiguity arising from language instructions is provided in Section 2.2, including the concept of clarifying questions and user preference as a disambiguation approach.

2.1 Background

2.1.1 Reinforcement Learning

Reinforcement learning is a learning framework in which an agent interacts with its environment, by trial and error, and learns the optimal behaviour or decisions to maximise a numerical quantity [Sutton *et al.* 1998; Akalin and Loutfi 2021]. These problems can be modelled by Markov Decision Processes (MDPs) [Puterman 2014]. The MDP formally describes an environment and is defined by the tuple $\langle S, A, p, r, \gamma \rangle$, where:

- *S*: The state space; possible configurations of the environment i.e. all information available to the agent
- *A*: The action space; possible actions or decisions the agent can make in the environment
- $p(s'|s, a)$: The transition dynamics; probability of transitioning from state *s* to state s' given the action *a* taken by the agent
- $r(s, a; s')$: Reward function; numerical reward the agent receives after taking an action *a* and transitioning from state *s* to state s' bounded by $[r_{MIN}, r_{MAX}]$
- $\gamma \in [0, 1]$: Discount Factor; the importance of future rewards to the current state.

Formally MDPs are used to model decision-problems that preserve the Markov Property, in which the current state *s* contains all the relevant information for transitioning to state s' given an action *a*, and no information from preceding states is required [Sutton *et al.* 1998]. The objective of the agent is to perform actions that maximise the cumulative expected rewards it receives in order to optimally solve a given task. A policy π function defines the action *a* that should be

taken by the agent in a given state and the value function $V^{\pi}(s) = \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)|s_0 = s]$ defines the future cumulated rewards the agent can expect if it follows that policy π starting from state s. Similarly the Q-value function, $Q^{\pi}(s, a) := \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)|s_0 = s, a_0 = a]$, defines the future cumulated rewards the agent can expect when starting from state s, taking action a and then following that policy π. The optimal policy π^* to solve a given task is one that gives the greatest expected return at each state s, such that $V^{\pi^*}(s) = max_{\pi} V^{\pi}(s)$ for all $s \in S$; similarly $Q^{\pi^*}(s, a) = max_{\pi} Q^{\pi}(s, a)$ for all $(s, a) \in S \times A$ [Bellman 1957; Tasse *et al.* 2020].

RL has been applied to a variety of scenarios and domains within social robotics where user needs continuously change and incremental learning is required, discussed extensively in Akalin and Loutfi [2021]. To address the problem of ambiguity in HRI, we are interested in task performance driven methods (where the reward is based on the task performance of the robot) using a verbal communication medium to complete an instruction. Additionally in such cases it would be beneficial to leverage the compositionality of both language and object oriented goals which share sub-task components (e.g. colour), to assist in disambiguation, in conjunction with user preferences and the ability to ask clarifying questions.

2.1.1.1 Compositional Reinforcement Learning

Recently Tasse *et al.* [2020] described a Boolean task algebra for reinforcement learning, demonstrating the ability to compose learned value functions of base tasks to solve new tasks quickly. Base tasks are atomic goal specifications that can be combined to produce noval tasks Tasse *et al.* [2020]. This ability to solve new tasks from learned skills, with no further learning, is important for lifelong learning and the goal of artificial general intelligence [Tasse *et al.* 2020]. Their work formalises the Boolean algebra structure for the composition of learned tasks and introduces a goal-orientated value function (encoding all goals in the environment) to achieve zero-shot composition for any new task.

To formalise the notion of Boolean task algebra for shortest path problems with deterministic dynamics [Bertsekas and Tsitsiklis 1991], Tasse *et al.* [2020] consider a class M of related undiscounted ($\gamma = 1$) MDPs, with an absorbing goal set $G \subseteq S$. For all tasks in M, the following assumptions are defined:

1. The tasks share the same S (state space), A (action space) and p (transition dynamics), where p is deterministic.
2. The reward $r_{s,a}$ is constant across tasks for all non-terminal states, where the reward functions only differ on the goal set G between tasks.

This assumption aligns with HRI in which the environment, with multiple goals, remains the same but the goals we instruct the robot to achieve and their desirability vary.

From the above assumption, the conjunction [1] ($\wedge$), of a set of tasks $\mathcal{M}$ with $\mathcal{M}_u$, $\mathcal{M}_{\varnothing} \in \mathcal{M}$ is defined as:

[1] The work by Tasse *et al.* [2020] defines the conjunction, disjunction and negation, however we only focus on the conjunction of tasks for this research

$$\wedge : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$$

$$(M_1, M_2) \mapsto (S, A, p, r_{M_1 \wedge M_2}), \text{where } r_{M_1 \wedge M_2} : \begin{array}{l} S \times A \to \mathbb{R} \\ (s,a) \mapsto \min\{r_{M_1}(s,a), r_{M_2}(s,a)\} \end{array}$$

such that:

$$\begin{aligned} r_{\mathcal{M}_u} &: S \times A \to \mathbb{R}\ (s,a) \mapsto \max_{M \in \mathcal{M}} r_M(s,a) \\ r_{\mathcal{M}_\varnothing} &: S \times A \to \mathbb{R}\ (s,a) \mapsto \min_{M \in \mathcal{M}} r_M(s,a) \end{aligned} \tag{2.1}$$

A second sparseness assumption is included to ensure the tasks have a Boolean nature [Tasse *et al.* 2020]. The assumption states that for all tasks $\mathcal{M}$ there are only two possible terminal reward values for all (g,a) in $G \times A$, therefore $r(g,a) \in \{r_\varnothing, r_u\} \subset [r_{\min}, r_{\max}]$ with $r_\varnothing \leq r_u$. Placing the above assumptions on the set of tasks $\mathcal{M}$, new tasks can be created by composing existing tasks [Tasse *et al.* 2020].

2.1.1.2 Extended Value Functions

The reward and value functions are then extended to solve tasks specified by the Boolean algebra in order for the agent to learn the required value, of not only achieving the closest goal, but of all the goals [Tasse *et al.* 2020].

The extended reward function is given by:

$$(s,g,a) \mapsto \begin{cases} \bar{r}_{\min} & \text{if } g \neq s \in G \\ r(s,a) & \text{otherwise,} \end{cases} \tag{2.2}$$

where $\bar{r} : S \times G \times A \to \mathbb{R}$, $\bar{r}_{\min} \leq \min\{r_{\min}, (r_{\min} - r_{\max})D\}$ and D is the MDP diameter [Jaksch *et al.* 2010].

The extended Q-value function is defined as:

$$(s,g,a) \mapsto \bar{r}(s,g,a) + \int_S \bar{V}^{\bar{\pi}}(s',g)\rho_{(s,a)}(ds') \tag{2.3}$$

where $\bar{V}^{\bar{\pi}}(s,g) = \mathbb{E}_{\bar{\pi}}[\sum_{t=0}^{\infty} \bar{r}(s_t, g, a_t)]$. The extended reward function specifies that if a terminal state for a different task is encountered by the agent it receives the largest penalty defined, and the extended Q-value function learns to achieve all the goals using the 'task-dependant' reward functions [Tasse *et al.* 2020].

As in Equation 2.1, the conjunction for a set of optimal extended Q-value functions $\bar{\mathcal{Q}}^*$ of a set of tasks $\mathcal{M}$ adhering to the same assumptions with $\bar{\mathcal{Q}}^*_u$, $\bar{\mathcal{Q}}^*_\varnothing \in \bar{\mathcal{Q}}^*$ for $\mathcal{M}_u$, $\mathcal{M}_\varnothing \in \mathcal{M}$, can be defined as:

$$\wedge : \bar{\mathcal{Q}}^* \times \bar{\mathcal{Q}}^* \to \bar{\mathcal{Q}}^*$$

$$(\bar{\mathcal{Q}}^*_1, \bar{\mathcal{Q}}^*_2) \mapsto \bar{\mathcal{Q}}^*_1 \wedge \bar{\mathcal{Q}}^*_2, \text{where } \bar{\mathcal{Q}}^*_1 \wedge \bar{\mathcal{Q}}^*_2 : \begin{array}{l} S \times G \times A \to \mathbb{R} \\ (s,g,a) \mapsto \min\{\bar{\mathcal{Q}}^*_1(s,g,a), \bar{\mathcal{Q}}^*_2(s,g,a)\} \end{array} \tag{2.4}$$

Due to the Boolean Algebra, established over the tasks $\mathcal{M}$ and extended Q-value functions $\bar{\mathcal{Q}}^*$, Tasse *et al.* [2020] prove that the optimal value function for a given task can be immediately found if a task can be written down under the Boolean algebra. i.e. $\bar{\mathcal{Q}}^*_{M1 \wedge M2} = \bar{\mathcal{Q}}^*_{M1} \wedge \bar{\mathcal{Q}}^*_{M2}$,

where $M_1, M_2 \in \mathcal{M}$. The work was able to demonstrate that zero shot transfer is achievable through the composition of the learned goal-orientated base value functions using standard Boolean Algebra operators.

2.1.2 Transformer Based Language Models

Natural Language Processing (NLP) techniques can be used to process verbal task instructions given to a robot, and translate them to a command that the robot is able to perform [Shridhar *et al.* 2020]. One NLP technique used is Neural Machine Translation (NMT), commonly performed using a transformer model [Vaswani *et al.* 2017a] which comprises an encoder-decoder network architecture, based on attention mechanisms, mapping input sequences to output sequences (seq2seq). NMT models can be pre-trained on large datasets and leveraged for translating instructions in a specific domain [Raffel *et al.* 2019].

The transformer model was introduced in the research by Vaswani *et al.* [2017b]. The transformer is a sequence to sequence model that is widely used in NLP for a variety of tasks, including machine translation and text generation [Raffel *et al.* 2019; Zhang *et al.* 2023]. The model comprises of an encoder-decoder framework, where a text sequence is transformed into a fixed vector representation and used to generate a task-specific output sequence auto-regressively [Vaswani *et al.* 2017b; Raffel *et al.* 2019; Fan *et al.* 2020].

The transformer model makes use of 'scaled-dot-product' attention mechanisms [Vaswani *et al.* 2017b], also known as self-attention, which updates each token by a weighted score based on all other tokens and is used by the decoder to generate task-specific outputs [Vaswani *et al.* 2017b; Raffel *et al.* 2019]. Self-attention is computed using:

$$Attention(q, k, v) = softmax(\frac{qk^T}{\sqrt{d_k}})v \tag{2.5}$$

and takes as input three token representations, namely q to query information from other tokens, and a key-value k, v pair to respond to queries [Bahdanau *et al.* 2014; Vaswani *et al.* 2017b]. The dot-product of the query with all keys (of dimension d_k) is computed and divided by $\sqrt{d_k}$, the softmax function is then applied and the weights of the values (dimension d_v) are calculated [Vaswani *et al.* 2017b]. The attention function is computed simultaneously on a set of queries represented by the matrix Q and the matrices of the keys and values (K and V) [Vaswani *et al.* 2017b]. They further extend the self-attention mechanism to multi-head attention:

$$Multihead(Q, K, V) = concatenate(h_1, h_2, ..., h_i)W^H \tag{2.6}$$

where $h_i = Attention(Q_i, K_i, V_i)$ and the projection is a parameter matrix W^H [Vaswani *et al.* 2017b; Zhang *et al.* 2023], determining that it was advantageous to apply linear projections to the queries, keys, and values multiple times using different learned linear transformations [Vaswani *et al.* 2017b; Zhang *et al.* 2023]. Following this, they conduct the attention function in parallel on each of these transformed representations. In the implementation, the queries, keys, and values computed for a single-head attention are simply split into several components [Vaswani *et al.* 2017b]. This ensures that models employing one or multiple attention heads maintain uniform size; thus, multi-head attention does not lead to an increase in model dimensions [Vaswani *et al.* 2017b].

Multi-head attention facilitates the model's ability to simultaneously attend to information across various representation subspaces and positions. Conversely, with a single attention head, averaging impedes this capability [Vaswani *et al.* 2017b; Zhang *et al.* 2023].

The architecture of the transformer is shown in Figure 2.1. The encoder and decoder are comprised of the same type of sub-layers, namely multi-head attention and feed-forward layers (where a two layer Multi-Layer Perceptron is applied to each element)[Vaswani *et al.* 2017b;

Fan *et al.* 2020]. A residual connection [He *et al.* 2015] and layer normalisation [Ba *et al.* 2016] are performed after each sub-layer [Vaswani *et al.* 2017b; Fan *et al.* 2020].

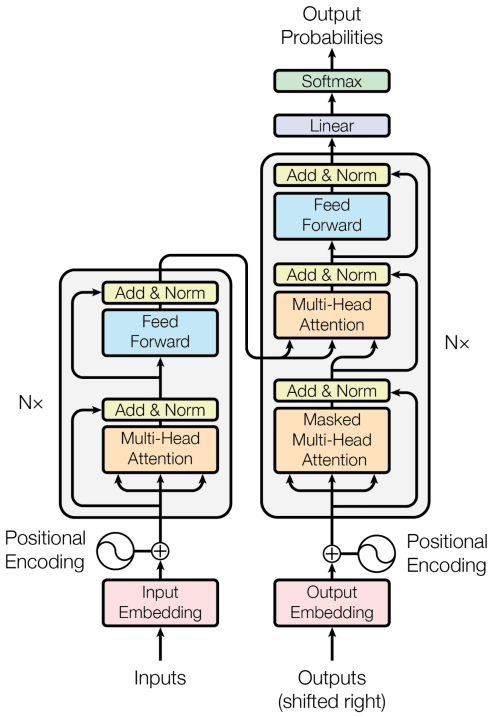


Figure 2.1: Transformer Architecture from Vaswani *et al.* [2017b].

These sub-layers are utilised to make what is called an encoder and decoder layer, which are then stacked to create N_x $(x = 6)$ identical layers respectively [Vaswani *et al.* 2017b]. A vector sequence is given as input to each layer and the output sequence is then passed to the next layer [Fan *et al.* 2020]. Positional encodings are utilised to ensure the model has access to the relative token positions of the encoder and decoder inputs [Vaswani *et al.* 2017b].

The output embedding (offset by one position) are input to the decoder, ensuring that model token predictions depend only on known output tokens at previous positions [Vaswani *et al.* 2017b], and the decoder output is passed to a linear transformation before being converted into the output token probabilities by a softmax function [Vaswani *et al.* 2017b].

Applications of the transformer based sequence-to-sequence model include generative pre-training on vast amounts of monolingual data in order to facilitate fine-tuning for down stream tasks (i.e. classification) [Raffel *et al.* 2019; Cohen *et al.* 2021], such as the T5 seq2seq transformer model, created by Raffel *et al.* [2019] to explore the effects of transfer learning, pre-trained on 750GB of web-corpra text.

2.2 Related Work

Having defined the background knowledge for our method we now expand on the related work specific to our approach. In this work we implement the approach by Cohen *et al.* [2021], who demonstrated the viability of using pre-trained transformer-based models, to map natural language instructions to learned base value functions. The designed framework leverages the compositionality of value functions and language, in an environment with goals that share components of their task descriptions. They make use of the Baby AI *PickUpObj* environment, which comprises various objects of different types and colours [Chevalier-Boisvert *et al.* 2018].

The agent learns atomic task-specific value functions, for manipulating objects defined by their attributes, which are then composed using the Boolean algebra to solve 18 novel tasks.

Their system makes use of the pre-trained T5 [Raffel *et al.* 2019] transformer model to translate referring expressions (i.e. "pick up the red ball") to the logical expressions (pickup_red and pickup_ball) of learned base tasks. The logical Boolean expressions, defined by the tokens representing the learned value functions and the logical operator, represent the composed goal value function. The T5 model is fine-tuned using RL, with a reward of +1.0 and -1.0 for the syntactically correct or incorrect Boolean output expression respectively. They found pre-trained T5 language model sufficiently generates the required Boolean expressions given natural language commands, and allows for a significant reduction in samples from the environment after the model learns to solve just a single task.

Their results showed that an agent can leverage existing learned object-attribute value functions to solve novel tasks efficiently, when given a natural language referring expression. The approach demonstrates, as in Mason and Lopes [2011], that learned tasks can be leveraged by a robot to complete an instruction. However, their solution is not able to handle the problem of ambiguity in the language instructions and in the environment.

2.2.1 Addressing Ambiguity

The requirement of social robots to understand instructions and learn to perform tasks accordingly has many challenges. One key challenge is that, while natural language provides a user-friendly interface to instruct a robot, it is also a source of ambiguity, especially when there is also ambiguity in the environment (i.e. similar objects to be retrieved), as in the example described in Chapter 1 where there are two possible cup options 'Glass and Porcelain' [Doğan 2021; Yang *et al.* 2022]. The main approaches, that can be used by collaborative robots to address verbal ambiguities, are asking for clarifications from the user, mapping user gesture or using user preference learned from previous interactions [Doğan 2021]. The ability for the robot to ask a question or consult known preferences helps with task efficiency and accuracy, rather than waiting for negative feedback after the task has been performed [Whitney *et al.* 2017].

The robot first needs to understand the referring expression and leverage it to locate the objects that match the given descriptor, either using visual sensors with object-detection [Doğan 2021] or from learned interactions. Based on this knowledge, if there are multiple objects that match the descriptor, the robot must identify the ambiguity and use the above approaches to complete the referring expression and disambiguate the request [Doğan 2021].

2.2.1.1 Clarifying Questions

Doğan [2023] created a novel system for generating clarifying questions to resolve ambiguities, using the object-attribute information from the user request (e.g. object type or colour) and a visual object detection framework. Their research was done in response to the defined shortfall of accurately identifying reference objects in an environment. They present their own pipeline using an image captioning module [Selvaraju *et al.* 2017] to identify regions containing the desired object-attribute (determined from parsing the referring expression) from RGB images of the environment with depth features (RGB-D). They then generate clarifying questions to resolve ambiguity if there are multiple active regions, using a learning-based approach that spatially refers to an object while taking the user's perspective into account (e.g. "Is it the blue ball to the left of the box?"). They evaluated their method (using 63 participants) against a baseline of simply asking the user to redescribe the object, determining that asking clarifying questions increases the efficiency of identifying correct objects and positively impacts user experience.

Another approach using clarifying questions, to infer the correct item given uncertain language instructions and user gesture, was evaluated by Whitney *et al.* [2017], where a user

instructed the robot to fetch a desired object while pointing to it. Their algorithm differs from Doğan *et al.* [2022] as it assumes a known list of fixed objects and makes use of user gesture rather than perspective. The delivery domain is formalised as a Partially Observable Markov Decision Process (POMDP) defined by the observation space, which models the uncertainty given the instruction and observed gesture, and determines whether or not to ask a point-based question (e.g. "This one?"). They compared the disambiguation time and accuracy, to similar baseline models, when the robot asked no questions [Whitney *et al.* 2016], asked questions until it achieved 95% certainty [Young *et al.* 2010] or utilised their intelligent "question-asking policy". They demonstrated that the intelligent asking policy improved accuracy and efficiency in real-word user evaluations, showing that too many clarifying questions can cause speech translation failures in addition to being unnecessarily exhaustive for the user.

Yang *et al.* [2022] used a combined approach of the above methods, making use of an object-detector (using RGB-D images) and an attribute-guided POMDP system, to determine whether to ask attribute-based or point-based clarifying questions to resolve ambiguity. The system asked an average of 1.71 questions in ambiguous settings and demonstrated that the ability to ask attribute-based clarifying questions results in accurate disambiguation in 91.43% of real-robot experiments. They compared their system with that of Whitney *et al.* [2017] (which only asks point-based questions) showing notable increases in accuracy and decreases in disambiguation time, determining that object attributes provide critical disambiguation information.

The above approaches show the promising results of using clarifying questions to resolve ambiguities, and provide various findings that can guide future research. However a notable shortfall in such methods is that they are not personalised to the user and do not consider user preference as a factor to assist disambiguation, solely focusing on determining the type of question to ask or whether to ask a question. By not 'remembering' the choice a specific user made in previous interactions, they cannot leverage this information when faced with similar instructions and previously resolved ambiguities. They also rely on user perspective or gesture, assuming the user is present in the same scene, these use cases will not extend to a domain where the robot is instructed by the user from another room. The systems also only make use of "off-the-shelf" object detectors or defined object lists, which do not generalise well to complex real-world environments and do not leverage knowledge learned from previous interactions with the environment.

2.2.1.2 User Preference

User personalisation in HRI allows the needs or preferences of an individual to be considered during the interaction; such personalisation increases user trust in the system [Lee *et al.* 2012a] and makes the interaction easier [Hellou *et al.* 2021]. Lee *et al.* [2012a] and Lee *et al.* [2012b] created a personalised robot that delivered snacks to a user, the system kept a memory of the user's preferences, number of interactions with the user as well as its own behaviour. Their system was limited in that the robot speech was scripted by the operators but demonstrates the increase in cooperation received from users and the possibilities of engagement that can be achieved with personalisation.

Work aligned to the ambiguity problem in HRI, was done by Mason and Lopes [2011] who designed a dynamic model that receives verbal instructions from a user and learns individual preferences to select an appropriate task autonomously. Their system creates a user profile, by learning the user classification of the world state (good/bad) after executing an instructed task. This profile is then used by the robot to select its own goals in order to get the environment to a 'good' state (e.g tidying a set of objects). They encode the geometry of the environment and locations of objects (with their properties and relations) as a set of features, and define a dataset of feature label pairs, capturing if the user considers a set of features to be a good or bad world state. A beta classifier is used to generalise the user preference information over the feature space. The robot must then determine a plan to reach a more desirable state. The robot is not

able to rely on clarifying questions, when user commands are given. It therefore deals with incomplete language instructions by using the planner to simply take a step in the environment, that will move the world into a more optimal state. In their domain they were able to show that 80-130 interactions with a user is sufficient to acquire a usable preference profile. This approach also demonstrates learned tasks can be leveraged by a social robot to achieve a goal, but the environment presented is limited as it requires user labeled data regarding the world state and a feature vector to be continuously updated when the environment changes.

Similar research shows the benefit of using user preference in social robots [Hellou *et al.* 2021], but no work has been found where user preference is employed specifically to address ambiguity in conjunction with clarifying questions. The use of both clarifying questions and preference can help make robots more efficient and adaptable, in cases where the user is not in the same scene, and prevent repeated unnecessary clarifying interactions (i.e. if the robot knows that the user statistically prefers red balls over blue balls it does not have to clarify the ambiguity every time this scenario occurs). Additionally it is noted that the approaches studied to address ambiguity will not scale adequately to real-world non-stationary settings with complex environments and do not leverage learned base skills and the life-long learning capabilities of RL agents [Parisi; Akalin and Loutfi 2021]. Ideally we would like the agent to not only learn user preferences and have clarification abilities, but also leverage existing knowledge it has from interacting with its environment (i.e. object-based learned skills). We present a framework to address these shortfalls in Chapter 3.

Chapter 3

Research Methodology

This chapter outlines the research methodology for the various components of our proposed framework. We define our approach to address ambiguity in HRI (resulting from natural language instructions), that leverages the compositionality of learned object-attribute base-tasks in conjunction with user preference and clarifying questions for adaptive task inference. The system design is guided by the background knowledge and shortfalls seen in the related work outlined in the previous chapter. The first part of the research consists of training and testing compositional agents that follow language commands outlined in Section 3.2. In Section 3.3 we define our method for adaptively inferring the complete task instruction, by leveraging learned base-tasks when ambiguity is introduced. The framework includes the implementation of a question and answering system to clarify ambiguity, outlined in Section 3.3.1; and the application of dynamic user preferences (Section 3.3.2) to prevent unnecessary repeated clarifying interactions and the need for the user to be physically present in the same scene e.g. to point to the required object. To explain our methodology, we consider an example domain where a robot is in a room with a number of objects that share components of their semantic properties (i.e. type, material or colour) and a human gives an ambiguous referring instruction to have one of the objects picked up by the robot, as in Figure 1.1.

3.1 Overview

An overview of the complete system is defined in Algorithm 1, where the input is defined as the natural language command from the user and the output is the execution of the task by the agent. Once an instruction has been given, the algorithm determines if the referring expression is ambiguous given the observable environment. If there is ambiguity it will attempt to clarify the ambiguity using known user preferences. Should there exist uncertainty in the user preferences the agent also has the option to ask a clarifying question. Once the ambiguity is clarified, the known compositional value functions of learned base-tasks are composed, and the inferred complete task is executed in the environment. The user preferences are iteratively updated after each interaction to increase task inference accuracy over time and reduce the number of required clarifying questions.

3.2 Compositional Agents

A noted impracticality of reinforcement learning approaches in HRI is the long training times required [Akalin and Loutfi 2021]. This can be exhausting for human users and cause considerable wear to robot hardware. As such, training and testing is generally performed in a simulated environment, where multiple algorithm variations can be validated before deploying the soft-

Algorithm 1: System Overview

Input: Natural Language Command from User
Output: Execution of Task by Agent
Data: Compositional value functions of learned base-tasks

```
/* Determine if the language command is ambiguous given the observable environment
   (Section 3.3)                                                                  */
if Language Command is Ambiguous then
    Consult user object-attribute preferences
    /* User preferences stored from previous interactions (Section 3.3.2)        */
    if Ambiguity can be resolved from known preferences then
        Determine desired object given the ambiguous command and user preferences
        Compose the compositional value functions of base-tasks
    /* Clarifying questions inferred logically from learned base-tasks (Section 3.3.1) */
    else
        Ask a Clarifying Question to resolve the ambiguity
        Compose the compositional value functions of base-tasks
else if Language Command is unambiguous then
    Compose the compositional value functions of base-tasks
Execute command
Update User Preferences
```

ware on a physical robot [Akalin and Loutfi 2021]. For our methodology we train compositional agents that follow language commands in a simulated environment. To achieve this we replicate the approach implemented in Cohen *et al.* [2021], learning the compositional value functions of the object semantic base-tasks in the environment, mathematically defined in Section 2.1.1. This approach provides sample-efficient agents that can use existing knowledge from interacting with the environment (i.e. object-based learned skills) such as "picking up a cup" and "picking up porcelain objects" to solve novel tasks [Tasse *et al.* 2020] e.g. "picking up a porcelain cup", as described in Chapter 1.

A Boolean label is assigned to each of the base-task compositional value functions we would like the agent to learn. The base-tasks are determined by selecting the minimal set of tasks that can be composed to produce the goal set of interest [Tasse *et al.* 2020], and correspond to an object's semantic attributes.

We represent each compositional value function $\bar{\mathcal{Q}}^*$ as a set of standard value functions, one for each goal $g \in G$ [Tasse *et al.* 2020]. As in Tasse *et al.* [2020] and Cohen *et al.* [2021] the Q-value functions for each goal are trained using deep Q-learning [Mnih *et al.* 2015] (i.e. a separate Deep Q-Network (DQN) is used to approximate the Q-value function for each goal g in the goal list $|\mathcal{G}|$). Deep Q-learning makes use of a neural network to approximate the Q-value function in environments with large action and or state spaces, and architecture and hyperparameter settings are outlined in the Appendix A [Cohen *et al.* 2021; Mnih *et al.* 2015]. The goal-oriented rewards, adhere to the spareness assumption required for the Boolean task algebra [Tasse *et al.* 2020], seen in Equation 2.2.

Considering an example in which an agent has learned the Q-value function for each object semantic task separately ($\bar{\mathcal{Q}}^*_{M_1}$) and ($\bar{\mathcal{Q}}^*_{M_2}$), it is possible to compose the learned value functions to solve tasks defined by their intersection [Tasse *et al.* 2020], shown in Equation 3.1.

$$\bar{Q}^*_{M_1 \wedge M_2} = \bar{Q}^*_{M_1} \wedge \bar{Q}^*_{M_2} := min\{\bar{Q}^*_{M_1}, \bar{Q}^*_{M_2}\} \tag{3.1}$$

Referring to our example domain described in Chapter 1, given an example instruction of picking up a porcelain cup (defined by the object's type and material attributes), we decompose

this into the Boolean expression describing the conjunction of base-tasks i.e. "pickup_cup AND pickup_porcelain" , whose respective learned value functions ($\bar{Q}^*_{cup}$) and ($\bar{Q}^*_{porcelain}$), can be composed to complete the required task in the environment:

$$\text{Example: } \bar{Q}^*_{cup \wedge porcelain} = \bar{Q}^*_{cup} \wedge \bar{Q}^*_{porcelain} := min\{\bar{Q}^*_{cup}, \bar{Q}^*_{porcelain}\}$$

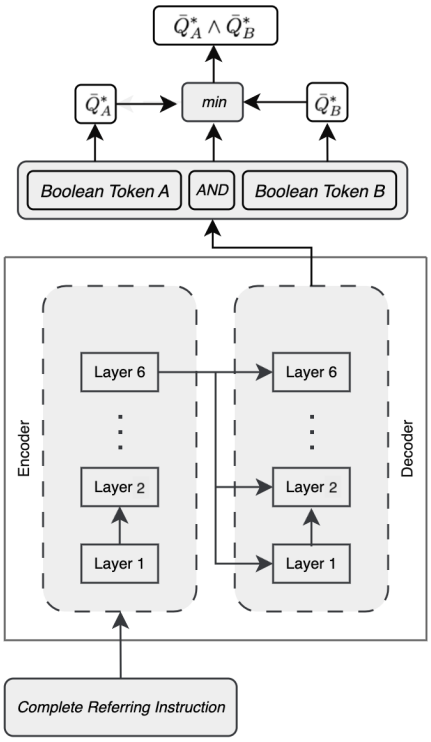
As defined in Tasse *et al.* [2020] the optimal action can be extracted from composed compositional value functions by selecting the maximising action after maximising over the goals to act in the environment ($\pi^*(s) \in \arg\max_{a \in A} \max_{g \in \mathcal{G}} \bar{Q}^*_{M_1 \wedge M_2}(s, g, a)$). Having learned compositional value functions of the object semantic base-tasks they can now be composed to solve new instructions without any further learning. This means that the agent does not have to learn every potential full instruction from the user, and is capable of solving an increasing number of instructions as the number of learned base-tasks increases [Tasse *et al.* 2020]. Additionally, these value functions can be leveraged to resolve ambiguity in the environment (explained in Section 3.3).

3.2.1 Grounded Language Learning

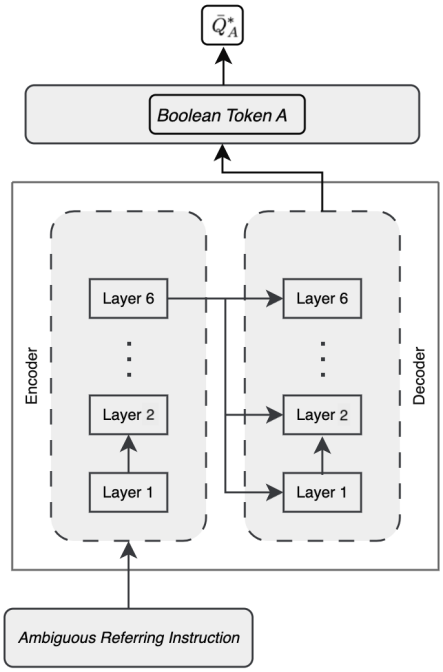
As in Cohen *et al.* [2021] we use a sequence-to-sequence machine translation model to convert from a natural language referring expression (e.g. "pick up the cup") to the logical Boolean tokens. The logical Boolean output expressions correspond to learned object semantic value functions used to solve the required task in the environment. The dynamic translations, using a NMT transformer-based model (described in Section 2.1.2), is implemented to prevent a non-expert user having to use specific predefined formal semantic language, when interacting with the agent, and also ensures that variances in the instructions do not result in a failure of the system.

To make the sequence-to-sequence model fit-for-purpose, we limit the vocabulary to the required Boolean labels of each of the semantic base-tasks e.g. "pickup_cup", the conjunction operator and end-of-sentence token $\{AND, < s >\}$ [Cohen *et al.* 2021]. As outlined in Cohen *et al.* [2021] the output is sampled from these defined tokens rather than the subword units to speed up training and ensure the output of valid tokens. While the defined vocabulary is fairly limited, the inclusion on the NMT allows for variations in the input referring expression that would not be viable in a hard-coded system. Temperature based sampling [Ackley *et al.* 1985] is used during training and greedy sampling during evaluation. The decoding is stopped similarly when the stop token is output or a defined number of vocabulary tokens are sampled (this is dependant on the number of object-attributes present in the environment) [Cohen *et al.* 2021].

The approached defined by Cohen *et al.* [2021] makes use of complete language instructions, seen in Figure 3.1a, which we adapt to include ambiguity in the instruction. When an ambiguous instruction is given to the system by the user (e.g. referring to an object by only one of its defining attributes), the incomplete instruction is translated by the NMT model and the output expression contains the token corresponding to the learned object-attribute value function, as shown in Figure 3.1b. This setup allows the user to provide a complete and incomplete referring expression. Rather than only outputting the complete Boolean expression that can be composed directly, we have to resolve any resulting ambiguity first. We also adjust for one word utterance inputs, to mimic a user answer to a question posed by the agent e.g. Question: "Which cup would you like", Response: "Porcelain". For our research we do not explicitly investigate the general case where a language instruction includes picking any object, i.e. "pick up *a* cup" and only test for specific instructions "pick up *the* cup", in order to test our disambiguation method.



(a) The intersection of the value functions is computed from an unambiguous referring instruction e.g. "pick up the porcelain cup', resulting in the composition of base-task value functions for completing the specific task in the environment.



(b) Ambiguity is introduced in the referring instruction e.g. "pick up the cup', mapping to a compositional value function that could be representative of a number of objects in the environment.

Figure 3.1: The NMT model translates referring instructions into Boolean expressions, representing compositional value functions.

3.3 Addressing Ambiguity

We propose a modular approach to address uncertainty arising from non-specific referring expressions, and to determine the correct object, should there exist multiple objects in the environment that have similar semantic properties. We leverage the learned semantic compositional value functions to identify objects in the environment matching the given referring expression. Should there exist multiple identified objects, the agent consults the users preferences over the object semantic properties to determine the correct object to act on and can ask informed clarifying questions if uncertainty exists.

If an incomplete referring expression is given to the agent, it is first decomposed into its corresponding Boolean token/s. Then, given the observable environment and the learned optimal compositional value functions for all tasks $\bar{Q}^*$, we retrieve the corresponding compositional value function $\bar{Q}^*_M$ from the Boolean token/s. We then maximise over the actions to recover the set of optimal value functions $Q^*_M(s,a)$ for the goals which are satisfiable ($G_{\text{satisfiable}}(s)$) given the current environment [Tasse *et al.* 2020], see Equation 3.2.

$$G_{\text{satisfiable}}(s) \coloneqq \{g \in G | \max_a Q^*(s,g,a) > 0\} \tag{3.2}$$

These goals correspond to objects present that share the semantic task attribute, which in the case of multiple objects with same attribute, results in multiple candidates. In cases where there

is only one optimal value function recovered, no ambiguity exists (i.e. there is only one goal object corresponding to the incomplete referring expression) and the system simply executes the base-task without determining the complete object-attribute description. Similarly, if there are identical objects, the system can also execute the base-task without determining the complete object-attribute description, since picking up either object will satisfy the goal.

In cases where there is more than one optimal value function recovered, ambiguity exists. Since we know the task M corresponding to the incomplete referring expression, and the set of solvable goals i.e. objects in the environment that share the task attribute, we can infer which attribute needs to be clarified in order to determine the complete task description. The modules to ask a clarifying question (See Section 3.3.1) or consult known user attribute preferences (See Section 3.3.2) are used to determine the correct object the user would like the agent to act on. Once this is determined the base-task compositional value functions can be composed, using the Boolean task algebra, to perform the complete task without further learning [Tasse *et al.* 2020]. The compositional semantic base-task approach allows ambiguity to be introduced in the referring expression, at the base-task level for all learned semantic attributes of an object, and provides the means for performing a complete task instruction without having to learn all possible combinations of value functions.

3.3.1 Clarification using Question and Answering

A question can be asked to clarify the instruction if there exists ambiguity in the environment. The proposed method, outlined in Algorithm 2, includes asking a clarifying question to resolve the ambiguity, composing the value functions and solving for the task. Considering our example shown in Figure 1.1 with the instruction: "Pick up the cup", we map the corresponding Boolean token "*pickup_cup*" to the compositional value function $\bar{Q}^*_{cup}$ (or more generally $\bar{Q}^*_{M_1}$) corresponding to the incomplete referring expression for picking up a cup.

We recover the set of value functions $Q^*_{M_1}(s,a)$ for the goals which are solvable given the environment $G_{\text{satisfiable}}(s)$, defined in Equation 3.2. If the number of satisfiable goals $|G_{\text{satisfiable}}(s)|$ is greater than one there exists more than one goal object corresponding to the incomplete referring expression i.e. A glass and a porcelain cup.

Since we know the task M_1 (i.e. "pickup_cup") we infer which object attribute needs to be clarified and ask the question i.e. Question: "Which material would you like?". The question contains the identified object-attribute task descriptors needed to resolve the ambiguity and is kept simple to prevent language failures at this stage. The user response i.e. Response: "Porcelain", is given and matched to the corresponding learned compositional value function $\bar{Q}^*_{porcelain}$ (or more generally $\bar{Q}^*_{M_2}$). The algorithm then composes the base-tasks to act in the environment $\bar{Q}^*_{M_1 \wedge M_2} := min\{\bar{Q}^*_{M_1}, \bar{Q}^*_{M_2}\}$. In the case where objects have more than two semantic properties, and should there still exist more than one goal object corresponding to the composed base-tasks i.e. a blue and a red porcelain cup, the algorithm allows for additional clarifying questions to be asked to resolve the ambiguity i.e. Question: "Which colour would you like?".

If an unambiguous expression is given, the output Boolean tokens correspond to the conjunction of the compositional value functions, as shown in Figure 3.1 A. Only one candidate would arise in this case from maximising over the actions of composed compositional value functions as it can only be satisfied by one object, and the system can execute the instruction directly.

3.3.2 Inference from User Preferences

The method outlined in the previous section represents an extreme case of always asking the user a clarification question, this can cause exhausting interactions between the user and agent resulting in an unusable system. To prevent this and allow the agent to learn from user interactions overtime, we implement user preference tracking based on the relative frequency over the

Algorithm 2: Algorithm for asking task-specific clarifying questions upfront to resolve the ambiguity and execute the task

Input: Compositional value function/s $\bar{Q}^*_{M_1}$ corresponding to the incomplete referring expression

Output: Composed compositional value function/s for the resolved instruction

Data:

- The set of optimal value functions $Q^*_M(s,a)$ for the goals which are solvable given the current environment

Compute: $G_{\text{satisfiable}}(s) := \{g \in G | \max_a \bar{Q}^*_{M_1}(s,g,a) > 0\}$

`/* Ask a task-specific clarifying question */`

while $|G_{satisfiable}(s)| > 1$ **do**

  Ask the user to clarify the required object attribute. (Q: "Which material would you like? / Q: "Which colour would you like?")

  `/* Parse the user response */`

  Map the user response to a compositional value function $\bar{Q}^*_{M_2}$

  `/* Compose the compositional value functions */`

  Compute: The intersection of the value functions $\bar{Q}^*_{M_1 \wedge M_2} := min\{\bar{Q}^*_{M_1}, \bar{Q}^*_{M_2}\}$

  `/* In the case where objects have more than two semantic properties */`

  Set: $\bar{Q}^*_{M_1} = \bar{Q}^*_{M_1 \wedge M_2}$

  Compute: $G_{\text{satisfiable}}(s) := \{g \in G | \max_a \bar{Q}^*_{M_1}(s,g,a) > 0\}$

else

  Language Command is unambiguous

number of interactions between the agent and the user, for each of the object semantic properties. The agent learns the preference profile for each user, across all object semantic attributes, which correspond to the learned base-task value functions. The probability $P(x_t)$ of a user preferring an object attribute x_t is based on the relative frequency $f(x_t)$ that the user selects objects with that attribute during the number of interactions N_t. The user preference probability for all object-attributes are recalculated after each interaction with the user, whether there is ambiguity in the command or not. The recalculation is done by increasing the number of interactions $\{N_t\}$ and the respective frequency count $\{f(x_t)\}$ (for the selected attribute) by one, and then calculating the probability (see Equation 3.3) for all attributes respectively:

$$P(x_t) = f(x_t)/N_t \tag{3.3}$$

Each user performs an initialisation step with the agent, in order to create an initial baseline user preference profile for all semantic object attributes $P(x_t)$. The number of initial interactions N_i required by the agent to learn a relatively accurate user profile is determined experimentally in Chapter 4. Following this initialisation step the user preference probabilities are continuously updated after each interaction with the user, to account for changing preferences over time. An initialisation step is chosen over the use of direct online learning as it is hypothesised that the user experience would suffer significantly from poor initial recommendations, however investigating this claim is left for future work outlined in Chapter 5.

As described in Section 3.3, given an ambiguous instruction and having determined the compositional value function $\bar{\mathcal{Q}}^*_{cup}$ (or more generally $\bar{Q}^*_{M_1}$) corresponding to the incomplete referring expression, we can infer which object attribute needs to be clarified to resolve the ambiguity, should it exist i.e. "Which material does the user prefer?". The user preference distribution is then consulted for the object attribute accordingly. From the distribution the most probable semantic attribute is determined. Assuming the preference probability for a given

attribute is higher i.e. $\bar{Q}^*_{porcelain}$ (or more generally $\bar{Q}^*_{M_2}$), the system will compose the base-tasks $\bar{Q}^*_{M_1 \wedge M_2} := min\{\bar{Q}^*_{M_1}, \bar{Q}^*_{M_2}\}$, act in the environment and the user preferences will be updated in line with Equation 3.3.

Following this, we combine the ability for the agent to track user preference and ask clarifying questions. Where an upfront clarifying question will only be asked if the user preference, between compared object semantic attributes, is not certain (e.g. if the preference probability is equivalent for object attributes or if the difference in probability is less than a predefined threshold). The question contains the identified object-attribute task descriptors needed to resolve the ambiguity as described in Section 3.3.1. The user response is given and matched to the corresponding learned value function, according to Algorithm 2, the resulting base-tasks are then composed to complete the required task in the environment. Should the most probable semantic attribute prove incorrect, resulting in the agent picking up the wrong object, the agent is able to correct the mistake by dropping the object and asking a clarifying question to resolve the ambiguity.

In Section 4.2.2 we investigate the limits across the various settings, namely using user preferences to infer the complete task with and without the ability to ask a clarifying question. We compare this setup to that of always asking a clarifying question upfront and empirically determine the best method based on accuracy, environment interactions and the user experience-cost of answering a question.

Chapter 4

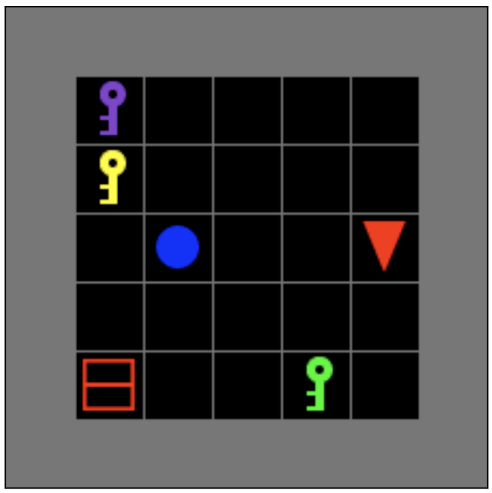
Experimentation

This chapter comprises of the experimentation performed to exhaustively test the framework described in the previous chapter. The framework includes training and testing compositional agents that follow language commands, and adaptively inferring task instructions, by utilising clarifying questions and the application of dynamic user preferences, should ambiguity exist. We start by outlining the environment used in Section 4.1. Once the environment has been defined, we introduce the baseline setup for comparing the results in Section 4.1.1. A modular approach is used to assess the performance of the system when resolving ambiguity, and comprehensive testing is done on the question and user preference components, with the results provided in Section 4.2.1 and 4.2.2 respectively. We then extend the proposed system to include the natural language commands, showing how this affects the disambiguation accuracy in Section 4.3, and summarise our findings on the viability of our method in Section 4.4.

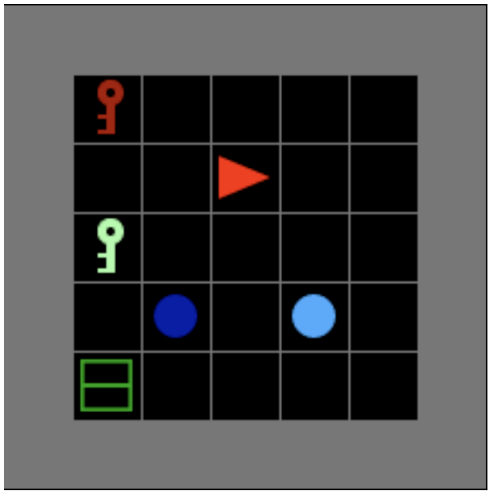
4.1 Environment

For this research, we exclusively test our hypothesis (that leveraging learned compositional base-tasks in conjunction with user preference and clarifying questions, is a viable approach to address ambiguity in HRI) in the 2D simulated BabyAI gridworld environment by Chevalier-Boisvert *et al.* [2018] also used in Cohen *et al.* [2021]. With this environment, exhaustive testing can be done on all aspects of the system and increased learning experience can be achieved when compared to real-world implementations. BabyAI is a common RL domain created for investigations in grounded language learning [Chevalier-Boisvert *et al.* 2018] and comprises 19 environment levels increasing in difficulty and instruction complexity. The domain allows for a fully observable environment, which aligns with having a physical robot including visual sensors to observe the environment configuration.

The environment contains various objects of different semantic attribute classes. These objects can be manipulated by the agent (i.e. picked up, dropped and opened). The agent is represented as a red triangle and can move in any principal compass direction. Each environment contains one goal object and the referring expressions are defined by the BabyAI "mission" statements (e.g. "pick up the blue ball") once a goal is set [Chevalier-Boisvert *et al.* 2018; Cohen *et al.* 2021]. The goal object attributes are set randomly for each episode in our respective experiments. Additional distractor objects are also introduced, creating ambiguity in the environment, which the agent is required to navigate around when given an instruction. For our experimentation, we make use of the 'pickup' task instruction and introduce ambiguity using two semantic object attributes (colour and type) respectively in a 5x5 gridworld domain shown in Figure 4.1a. In this two-attribute environment, the agent can navigate to and pick up objects described by three type attributes {*box, ball, key*} and six colour attributes {*red, blue, green, grey, purple, yellow*}. To test the scalability of our method, we further extend the environment to include objects with three semantic attributes, with the added complexity of ambiguity being



(a) Two-attribute environment:
Example 5x5 gridworld containing similar object types with different colour attributes. We see the similar colours and types introducing ambiguity into the environment.



(b) Three-attribute environment:
Example 5x5 gridworld containing similar object types with different colour and material attributes. With the various colour shades (dark and light) representing the metal and plastic objects respectively.

Figure 4.1: Two and three attribute BabyAI environments used to test the disambiguation methodology. These attributes result in eighteen possible navigation tasks in each environment setup. The agent is represented by the red triangle.

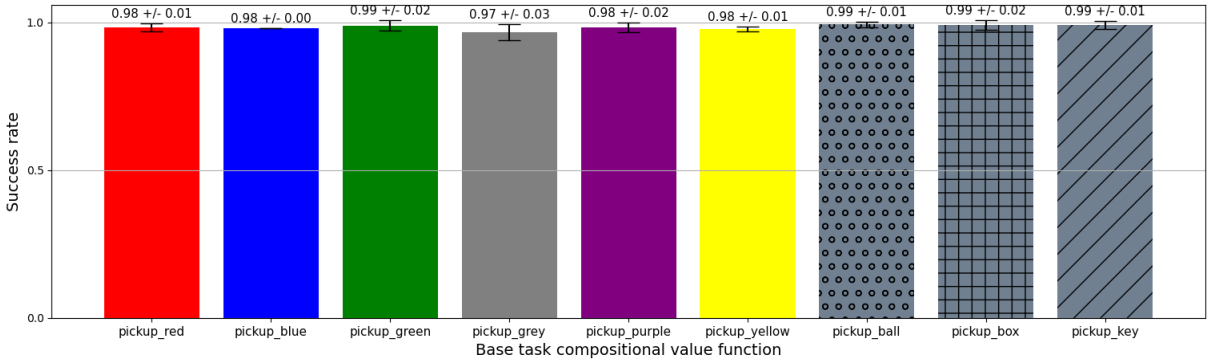
introduced in object colour, type or material shown in Figure 4.1b. In the three-attribute environment, the agent can navigate to and pick up objects described by three type attributes {*box, ball, key*}, three color attributes {*red, blue, green*} and two material attributes {*metal, plastic*}. We visually distinguish between the different object materials (metal and plastic) by varying the color of the object i.e. a red metal object will be a darker shade of red than a red plastic object. These attributes yield eighteen possible navigation tasks in each environment setup respectively.

4.1.1 Setup

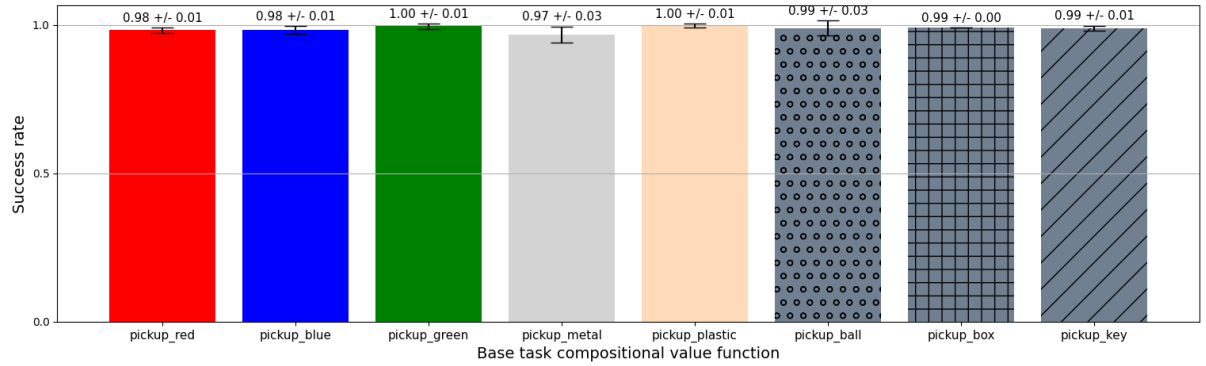
As a first step we train compositional agents that follow language commands. To achieve this we make use of the approach implemented in Cohen *et al.* [2021], explained in Section 3.2. This aligned implementation, in the same environment, also provides a comparable evaluation baseline which can be used to determine how well our system performs when the instruction is ambiguous, given the environment, compared to an unambiguous complete instruction.

As described in Section 4.1, the BabyAI environment allows for grounded language learning i.e. specifying a mission as "pick up the blue ball" or "pick up the metal blue ball". We decompose this into an algebraic expression describing the required atomic base tasks i.e. (*pickup_blue and pickup_ball*) or (*pickup_metal and pickup_blue and pickup_ball*), whose tokens represent learned compositional value functions that can be composed to complete the required task in the environment. The compositional value functions, for picking up objects defined by their semantic attributes are represented by the following tokens {*Object Type: pickup_box, pickup_ball, pickup_key*}, {*Object Colour: pickup_red, pickup_blue, pickup_green, pickup_grey, pickup_purple, pickup_yellow*} for the two-attribute environment. The three-attribute environment comprises the tokens {*Object Type: pickup_box, pickup_ball, pickup_key*}, {*Object Material: pickup_plastic, pickup_metal*}, {*Object Colour: pickup_red, pickup_green, pickup_blue*}.

For each mission, the training environment contains a goal object and four distractor objects. The agent must navigate around the distractor objects to solve the task. For testing and



(a) Success Rate of the trained base-task value functions for the two-attribute environment.



(b) Success Rate of the trained base-task value functions for the three-attribute environment.

Figure 4.2: Success Rate of the base task value functions for each environment setup. The success rate, with four random distractor objects in the environment, is combined and averaged over 100 episodes, with the standard deviations over 10 runs

method viability we explicitly convert from natural language command to algebraic expression and implement the dynamic translation using a transformer model in subsequent experiments, this is done to ensure the logic is working efficiently and allow us to note any degradation in performance accuracy resulting from the introduction of the NMT on the complete system.

We implement the method described in Section 3.2, using deep Q-learning [Mnih *et al.* 2015] to learn the Q-value function for each atomic "pickup" task. A goal buffer (set of reached terminal states) and experience replay buffer are populated with 1000 steps of exploration before the agent starts training. A goal is sampled from the goal buffer during each training episode and an episode terminates when the agent picks up any object. The agent is trained to solve one object-orientated base task at a time and the training ends once a 98% task success rate is achieved over the last 100 episodes, to prevent the composition of sub-optimal policies. The transition dynamics are deterministic, and at each action the agent is given a reward +2.0 for picking up the correct object and a reward of -0.1 everywhere else (within the given time steps) [Tasse *et al.* 2020]. The state space is the fully observable RGB image and at each episode the agent's starting position in the environment is randomised.

The hyper-parameter settings (defined in Appendix A) including the reward specifications are kept the same as Cohen *et al.* [2021] to ensure comparability in the evaluation metrics, as described in Section 3.2. The agent learns the 9 base-task value functions (see Equation 2.3), for picking up the objects defined by their semantic attributes (e.g. ("pickup_red": $\bar{\mathcal{Q}}^*_R$),

("pickup_ball": $\bar{\mathcal{Q}}^*_B$)), which can then be composed using Boolean algebra to solve the 18 novel tasks in both the two-attribute and three-attribute environments (e.g. "picking up the red ball": $\bar{\mathcal{Q}}^*_R \wedge \bar{\mathcal{Q}}^*_B$). For our experiments all testing is limited to the intersection between policies. We show the success rate for each of the learned compositional value functions for the two and three attribute experiments in Figures 4.2a and 4.2b.

The results show a similar range [97% - 100%] for the success rate of each of the base task value functions tested in each environment configuration, with four random distractor objects.

4.1.2 Baseline Results

We define our baseline as the results obtained when a full unambiguous command is given by the user. An unambiguous command is defined as a mission that contains all the goal object's semantic properties i.e. "pickup the blue ball", required by the agent to complete the task, as opposed to an ambiguous command "pickup the blue object". This baseline definition provides a measure of comparison in terms of accuracy as well as the number of environment interactions required by the agent to solve the task. As in training, each environment setup has a goal object and four distractor objects. We adopt five experimental settings to determine the impact of ambiguous distractors on the performance. An ambiguous distractor is an object that shares a least one semantic attribute with the goal object e.g. "blue key" is an ambiguous distractor sharing a colour attribute with the goal "blue ball" object.

The five experimental settings are defined as follows:

1. Distractor objects sampled randomly (between zero and four ambiguous distractors sharing an object attribute with the goal object);
2. At least one ambiguous distractor sharing an object attribute with the goal object;
3. At least two ambiguous distractors sharing an object attribute with the goal object;
4. At least three ambiguous distractors sharing an object attribute with the goal object;
5. All four distractors sharing an object attribute with the goal object.

Accuracy As defined in Section 3.2 the full unambiguous command is decomposed into its corresponding Boolean expression mapping to the relevant base-task value functions, that are then composed to complete the required task in the environment. We provide baseline results for both the two-attribute and three-attribute environment settings to determine the accuracy variation when composing two or three compositional value functions. Ambiguity is introduced for each object attribute class (colour, type or material) separately. The accuracy of the agent across each of the five experimental settings for each shared object attribute are combined and averaged over 100 episodes with a maximum episode time-step of 20 and are provided in Table 4.1.

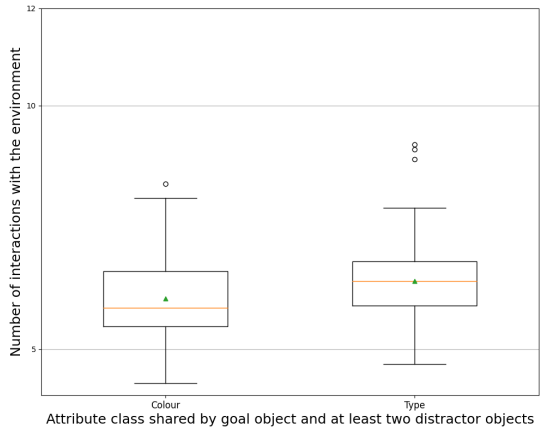
The RGB observation space allows the agent to generalise well, however we see that the accuracy decreases marginally as the number of distractors sharing an attribute with the goal object increases. The accuracy of the first experiment, where distractor objects are sampled randomly reflects the average performance of the experiments, lying between the accuracy seen when two or three distractor objects (sharing at least one object attribute) are present in an environment setup. We also see a small decrease in the success rate when compared to the individual base task value functions, seen Figure 4.2a and 4.2b , resulting from the composition of value functions. However, across the two environments the performance of the composed value functions are almost equivalent, meaning the added complexity of the three-attribute environment does not hinder performance when an unambiguous command is given to the agent.

No. of Ambiguous Distractor Objects	Avg. Accuracy (Two-Attributes)	Avg. Accuracy (Three-Attributes)
Random	96.98% ± 2.1%	96.94% ± 1.4%
1	97.52% ± 1.5%	97.54% ± 1.6%
2	97.43% ± 1.4%	97.08% ± 1.5%
3	96.85% ± 2.0%	96.83% ± 1.6%
4	96.75% ± 1.7%	96.67% ± 1.5%

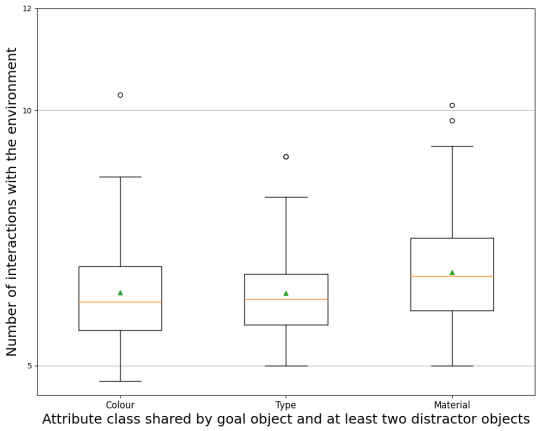
Table 4.1: Accuracy when a full unambiguous command is given by the user for each environment type with ambiguous distractor objects. The ambiguity is introduced across each of the object attributes separately. The results are combined and averaged over 100 episodes, with the standard deviations over 10 runs.

It is also observed that the variation in accuracy across experiments is not statistically significant, reinforcing that the agent can successfully complete the task, regardless of the number of distractors, when the language instruction is unambiguous.

Environment Interactions To measure the impact on performance when ambiguity is introduced for each object attribute class, we obtain the average number of interactions by the agent with an environment containing at least two ambiguous distractors of that class, representing the mid-range distractor setting. The number of interactions are defined as the amount of actions the agent takes before solving the task or the maximum allowed time-steps are reached. Figures 4.3a and 4.3b show the results for each attribute class and each environment setup respectively.



(a) Two-attribute Environment: The number of interactions with the environment for each ambiguous attribute class (*Colour and Type*) shared by the goal object and at least two distractors.



(b) Three-attribute Environment: The number of interactions with the environment for each ambiguous attribute class (*Colour, Type and Material*) shared by the goal object and at least two distractors.

Figure 4.3: Environment interactions across object attributes given that there exists at least two ambiguous distractor objects. Averaged over 100 episodes, for 10 runs.

For the two-attribute environment we see that the average number of interactions with the environment is 6.04 when ambiguity is introduced by distractors sharing the same *Colour*, compared to the same *Type* attribute at 6.39. The three attribute environment presents similar results with the average number of interactions being $6.43, 6.42, 6.83$ across *Colour, Type and Material* respectively.

4.2 Resolving Ambiguity

Having defined our baseline as the performance when an unambiguous command is given, we now test the methodology described in Section 3.3 to resolve ambiguity, when a non-specific command is given. Given an environment with a "red ball" and a "blue ball", an ambiguous command is defined as a mission that does not contain all of the goal object's semantic properties e.g. "pick up the ball". In this case the ambiguity around the goal object's colour needs to be resolved, in order for the agent to complete the task successfully .

We implement a modular approach to test the components of our methodology and address uncertainty arising from ambiguous referring expressions. The components include determining the performance when asking a clarifying question upfront (Q) to resolve ambiguity (i.e. 'Which colour would you like'), defined in Section 4.2.1, followed by the implementation of user preference tracking over the object semantic properties (i.e. 'Based on previous interactions I know which colour you would like'), and then finally the combination of user preference with the option to ask a clarifying question when the agent is uncertain (UP+Q), defined in Section 4.2.2.

The different components are compared to each other and the baseline across three specific criteria, 1) the accuracy of the algorithm, 2) the number of interactions by the agent with the environment and 3) the number of interactions with the user. The number of interactions with the environment is directly proportional to how quickly the ambiguity can be resolved and the number of interactions with the user should ideally be kept to a minimum to improve user experience, without compromising accuracy. For environment and user interaction experiments we only provide the results for the case where two ambiguous distractor objects exist, in order to compare efficacy for the mid-range distractor case across different settings.

During testing, the unambiguous ground truth instruction (e.g. "pick up the red ball") is generated randomly, converted into the Boolean expression and stored. This is done in order for system to be tested efficiently, with access to the full ground truth instruction. Having access to the ground truth also prevents exhaustive interactions with a user, which can slow down and hinder the viability testing of the system [Akalin and Loutfi 2021]. An ambiguous instruction is then given to the system (e.g. "pick up the ball"). Ambiguous distractors can be introduced at this point for any of the semantic properties i.e. type, colour and/or material depending on the environment. As with the baseline, each environment setup has a goal object and four distractor objects. The agent must navigate around the distractor objects of different attributes, resolve the ambiguity and use the intersection of the resulting value functions e.g. (*"pickup_red"* AND *"pickup_ball"*) to solve the task successfully. An episode terminates when the correct object is picked up or the maximum time-steps are reached. An incorrect object picked up by the agent is dropped and remains in the environment. The two-attribute environment is used to extensively analyse our method, determining the viability of each module before proceeding to the next. Once the algorithm has been analysed we implement it in the three-attribute environment, to test the scalability.

4.2.1 Question and Answering

A question can be asked by the agent to clarify the instruction if there exists ambiguity in the environment. The question contains the identified object-attribute class descriptor needed to disambiguate the command (e.g. Which colour would you like?" or "Which type would you like?" accordingly). To test the effectiveness of the clarification question and answering (Q&A) given by Algorithm 2 as defined in Section 3.3.1, we implement three experiments.

The experiments include:

- (R) A random execution, comprising of a naive approach in which the agent simply executes the ambiguous command in the environment, picking up objects that match the given ambiguous descriptor *"pickup_ball"* until the task is solved or the maximum time steps are reached. This allows us to determine a naive baseline of accuracy and number of interactions with the environment given a random algorithm.
- (R + Q) An initial random choice based on the ambiguous instruction, followed by a clarifying question if the object chosen by the agent is incorrect. This method presents the opportunity for the agent to select the right object initially by chance, without the user experience overhead of asking a clarifying question upfront.
- (Q) The last method implemented, outlined in Algorithm 2, includes asking clarifying questions upfront to resolve any ambiguity and solve for the complete task.

Accuracy The accuracy results for each experiment are shown in Table 4.2. In line with the baseline, all five distractor experiments setups are implemented, where the ambiguity is introduced across each of the object attribute classes separately and averaged over 100 episodes.

No. of Ambiguous Distractor Objects	**Avg. Accuracy (R)**	**Avg. Accuracy (R+Q)**	**Avg. Accuracy (Q)**	**Δ Baseline**
Random	94.78% $\pm$ 2.4%	95.22% $\pm$ 2.3%	**97.25**% $\pm$ 1.4%	+0.27%
1	92.44% $\pm$ 3.2%	93.52% $\pm$ 2.7%	**97.38**% $\pm$ 1.4%	- 0.14%
2	89.55% $\pm$ 3.5%	93.02% $\pm$ 2.5%	**97.20**% $\pm$ 1.4%	- 0.23%
3	88.25% $\pm$ 3.6%	92.03% $\pm$ 2.9%	**96.84**% $\pm$ 1.4%	- 0.01%
4	87.71% $\pm$ 5.5%	92.34% $\pm$ 2.0%	**96.22**% $\pm$ 1.7%	- 0.55%

Table 4.2: Two-attribute Environment: The accuracy for each experiment, when an ambiguous command is given by the user with various ambiguous distractor objects in the environment. The ambiguity is introduced across each of the object attributes separately, combined and averaged over 100 episodes. With the standard deviations over 10 runs.

As with the baseline accuracy results, we see that accuracy decreases marginally as the number of distractors sharing an attribute with the goal object increases, across all Q&A experiment settings. The random (R) setting provides the lowest accuracy. This is expected as the agent, retrying 1.72 times on average before successfully completing the task, does not always reach the goal object before the max time-step limit is reached. It is also noted that in cases where the number of retries is ≥ 3 the agent can get stuck in a position choosing a sub-optimal action repeatedly. We see that the variances in (Q) setting are much lower than that seen in the (R) and (R+Q) settings, meaning that it is far more reliable on average.

The results show that the accuracy of asking a clarifying question upfront (Q) is only marginally less accurate ($< 1\%$) than the easier baseline cases, where the agent is given the full unambiguous referring expression and is not required to resolve any ambiguity present. This demonstrates the viability of this method to resolve ambiguity. We see a marked jump in the performance of this algorithm in experiment (1) where distractor objects are sampled randomly, which is explained by the fact that in some cases there will not be ambiguous distractors in the environment for this experiment. In cases where the environment is unambiguous the agent can simply execute the individual base task value function without suffering losses resulting from the composition of value functions, resulting in a higher accuracy rate.

Environment Interactions We obtain the average number of interactions by the agent with an environment containing at least two ambiguous distractors of a class for each algorithm, and

show the results in Figure 4.4. Compared to the baseline results for the two-attribute environment setup, the average number of interactions, when using the (Q) approach of Algorithm 2, is 6.13 when ambiguity is introduced by distractors sharing the same *Colour*, compared to the same *Type* attribute at 6.65, which is a minimal 2.25% increase on average. These results show that the (Q) setting is able to perform similarly to the easier, unambiguous baseline.

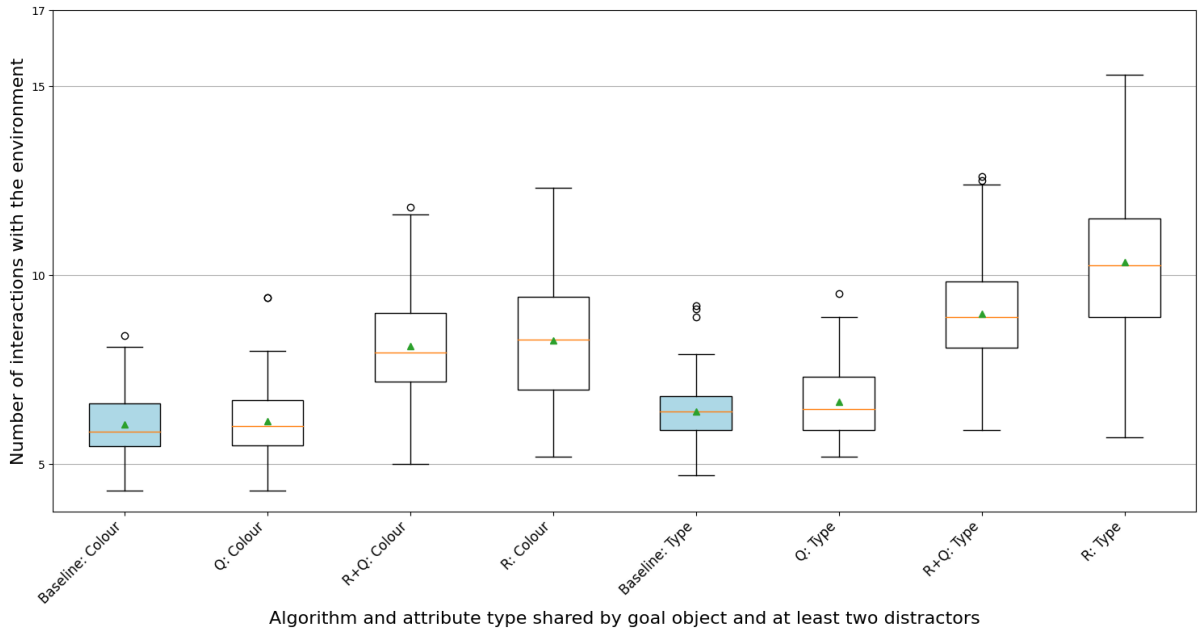


Figure 4.4: Two-attribute Environment: Average episode length for each object attribute class across various algorithms in an environment with two ambiguous distractors, averaged over 100 episodes for 10 runs. The baseline results are highlighted and represent the results seen in the simpler unambiguous case.

User Interactions The user interactions are defined as the number of questions the agent asks the user, to resolve the ambiguity. We obtain the average number of interactions by the agent with the user when there are at least two ambiguous distractors sharing an object attribute with the goal object. As seen in Figure 4.5 asking a clarifying question upfront (Q) results in the highest number of interactions with the user (~ 1 question per episode on average), resulting in the highest accuracy. In comparison to the baseline, the agent does not interact with the user when a full command is given. Simply executing the command in the environment to retrieve the required object.

Scalability To determine how well the (Q) approach of Algorithm 2 scales, we implement it in the three-attribute environment and obtain the accuracy (shown in Table 4.3), environment interactions (shown in Figure 4.6) and user interactions (shown in Figure 4.7).

No. of Ambiguous Distractors	**Accuracy (Q)**	Δ Baseline
2	96.60% $\pm$ 2.2%	-0.48%

Table 4.3: Three-Attribute Environment: The accuracy, when an ambiguous command is given by the user with two ambiguous distractors. The ambiguity is introduced across each of the object attributes separately, combined and averaged over 100 episodes. With the standard deviations over 10 runs.

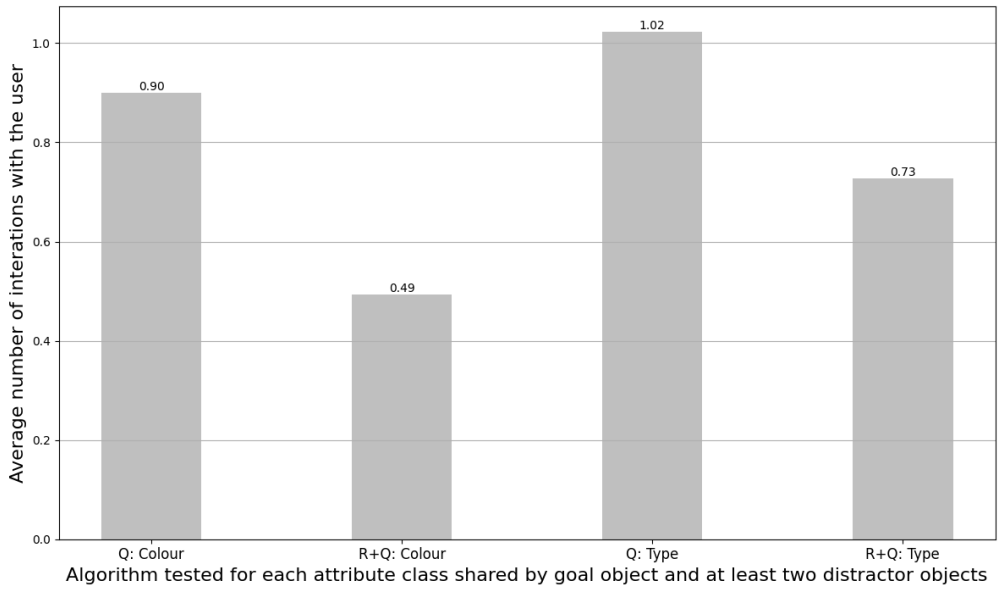


Figure 4.5: Two-attribute Environment: Average number of user interactions for each object attribute class across various algorithms in an environment with two ambiguous distractors, averaged over 100 episodes for 10 runs.

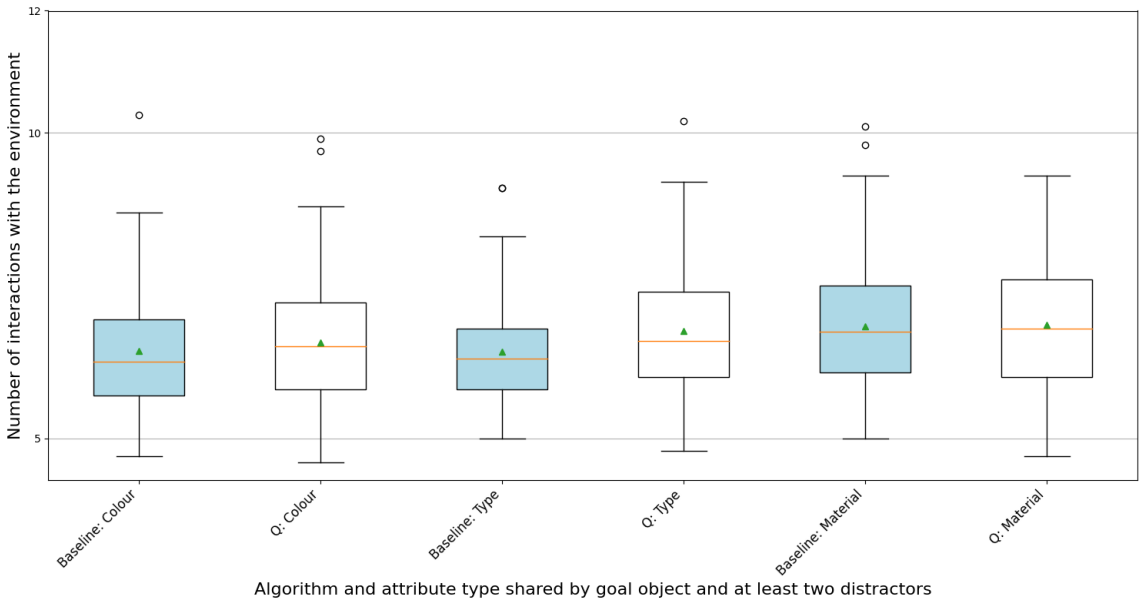


Figure 4.6: Three-Attribute Environment: Average episode length for each object attribute class across various algorithms in an environment with two ambiguous distractors, averaged over 100 episodes for 10 runs. The baseline results are highlighted and represent the results seen in the simpler unambiguous case.

The accuracy for the (Q) experimental setting is less accurate ($< 1\%$) than the easier unambiguous baseline result. While the average number of interactions, is $6.56, 6.76$ and 6.85 when ambiguity is introduced in the *Colour*, *Type* and *Material* class respectively, which is a minimal 2.24% increase on average, considering the increased complexity of the ambiguous setting. These results are directly in line with what we see in the two-attribute environment metrics. The largest increase is seen in the user interactions as the agent has to ask ~ 2 questions per

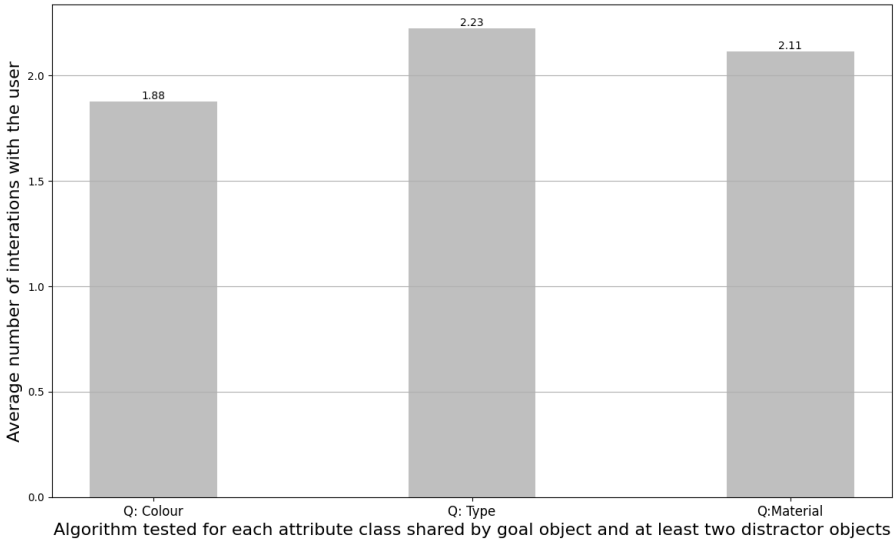


Figure 4.7: Three-Attribute Environment: Average number of user interactions for each object attribute class across various algorithms in an environment with two ambiguous distractors, averaged over 100 episodes for 10 runs.

episode on average, which is expected due to the number of object semantic attributes e.g. if the colour was specified "pick up the blue object", the agent would have to clarify the type "Which type would you like?" and if that is not enough to resolve the ambiguity, would follow up with "Which material would you like?". Overall we note that Algorithm 2 is able to scale sufficiently, demonstrating similar accuracy, when there is added complexity in the environment.

4.2.2 User Preference

The methods outlined in the previous section include the extremes of never asking the user a question (R) or always asking for clarification (Q) from the user. These methods can cause time intensive interactions between the user and agent. To overcome this and allow the agent to learn from user interactions over time, we implement user preference tracking. The agent learns the semantic object preference probabilities for each user, across all object attributes separately. The probability of a user preferring a semantic attribute is based on the relative frequency count of the user selecting a semantic property, over the number of interactions with the user, as outlined in Section 3.3.2. The ability to track user preference allows the agent to infer the required object, given ambiguity, without interacting with the user (i.e. "Based on previous interactions I know which you colour you would like"). This approach creates a more seamless user experience over time.

The initial probability $P(x_t)$ of a user preferring an object attribute x_t is uniformly distributed across all semantic attributes (see Equation 3.3), indicating no prior preference. The user then completes an initialisation setup in order for the agent to create a user preference profile for all object semantic attributes. Following this initialisation step, the agent has a baseline profile of the user which it can continuously update and use to complete subsequent instructions without the need to always ask clarifying questions. We experiment with the number of initial interactions required by the agent to learn a relatively accurate user profile. For this experiment we randomly generate the probability distributions (ground truth) of a user's preferences over the semantic attributes for each object class. We then investigate the average number of episodes required for the agent to learn a relatively accurate user profile.

An ambiguous command, sampled from the generated user preference distribution, is given

to the agent e.g. “pick up the ball”. The agent resolves ambiguity by determining which one of the satisfiable goals, see Section 3.3, has a higher preference probability. The determined base-task value functions are then composed to complete the required task in the environment. As with the previous experiment setup in Section 4.2.1 the episode terminates when the maximum time-steps are reached or when the correct object is picked up. The user profile is updated after each episode (as per the method defined in Section 3.3.2) and the agent then uses this profile to act in the next episode, given a new ambiguous instruction.

The experiment is run for each attribute class over 100 episodes, for 5 runs, in an environment with at least two ambiguous distractors. The probability distribution of a user's preferences is randomly generated at the beginning of each run. This is done to determine the optimal number of initial interactions over varying user preference distributions, to ensure generalisation across user profiles. We obtain the absolute mean probability error between the learned user profile distribution and the ground truth distribution per episode, seen in Figure 4.8. Based on the results, we set the number of initialisation steps N_i to be 60 episodes where the absolute mean probability error is ~ 0.20. Additionally, we see that the average number of attempts, required by the agent to select the correct object, drops by 34% from 2.6 attempts in the first episode to 1.7 attempts after the initial 60 episodes, and by 54% to an average of 1.2 attempts by the final episode, as shown in Figure 4.9.

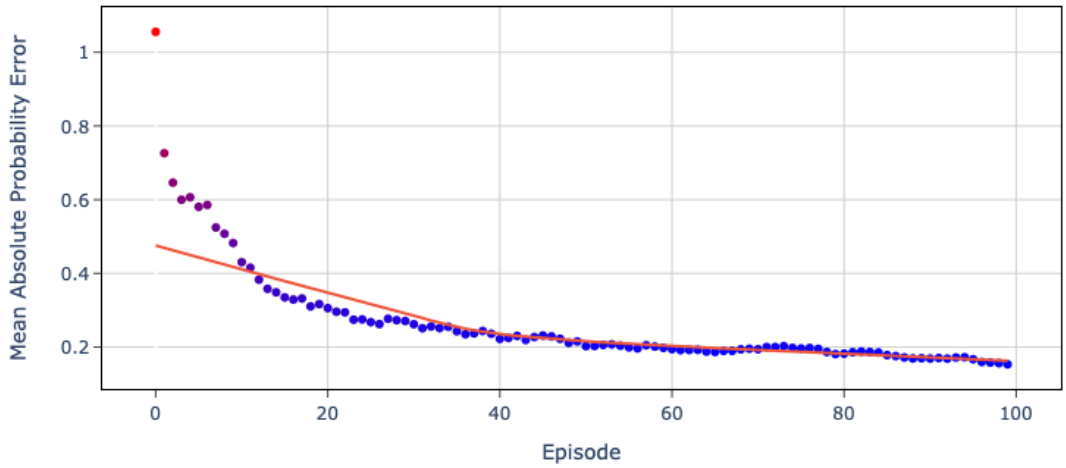


Figure 4.8: Two-attribute Environment: Mean absolute probability error between the learned user profile and the ground truth distribution. Ambiguity is introduced across each of the object attributes separately, and the results are combined and averaged over 100 episodes, across the 5 runs, in an environment with two ambiguous distractors. We emphasise the decreasing probability error as the number of episodes increases with the red trend line, which smooths the local variations in the data to highlight the overall downward trend.

This initialisation setup ensures the agent has learned an accurate initial user profile upfront, while limiting the initialisation time for improved user experience. The learned user profile compared to the ground truth distribution, following the initialisation setup for each object attribute class, is shown in Figures 4.10a and 4.10b. We see from these results that after 60 episodes the agent has sufficiently learned the user profile (We define the initialisation steps to be 60 for our experimentation going forward and leave a detailed ablation study for future work). Once the initialisation step is complete, we implement the ability for the agent to ask the user a clarifying question, defined in Section 3.3.1. A clarifying question will be asked if the user preference, between compared object semantic attributes is equivalent or if the difference in probability is less than a set threshold. We utilise the user profile learned from the initialisation setup to determine this uncertainty threshold.

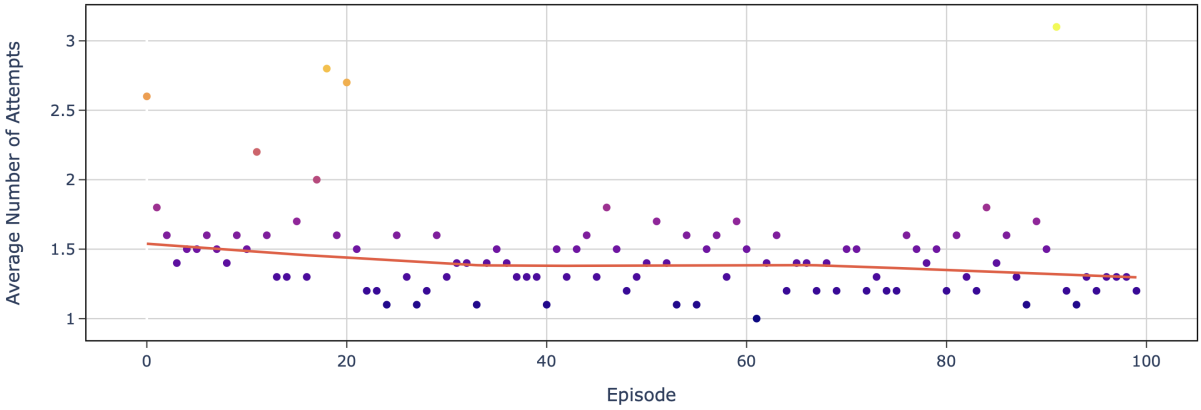
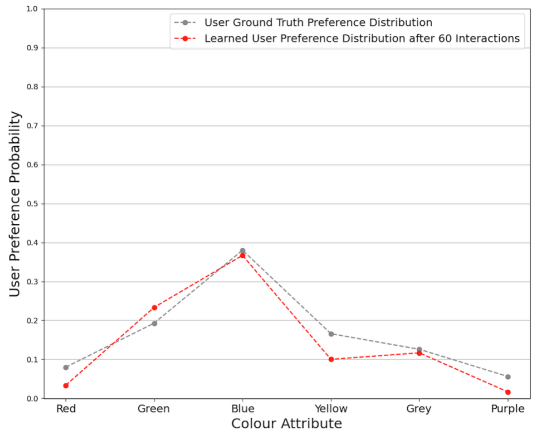
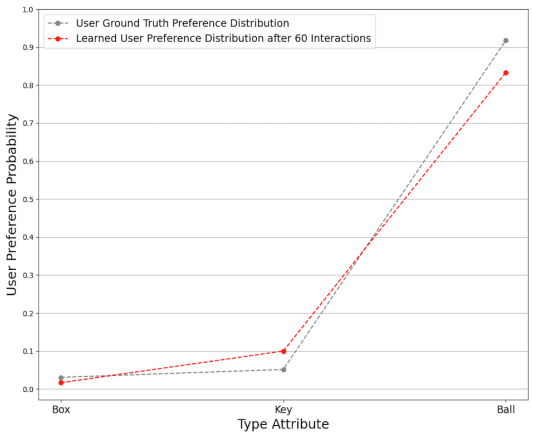


Figure 4.9: Two-attribute Environment: Average number of attempts required by the agent to select the correct object. Ambiguity is introduced across each of the object attributes separately, and the results are combined and averaged over 100 episodes, across the 5 runs, in an environment with two ambiguous distractors. We emphasize the decreasing number of attempts as the number of episodes increases with the red trend line, which smooths the local variations in the data to highlight the overall downward trend.



(a) Learned user profile for the Colour attributes after the initialisation setup, compared to the ground truth user distribution.



(b) Learned user profile for the Type attributes after the initialisation setup, compared to the ground truth user distribution.

Figure 4.10: Two-attribute environment: The learned user profile over each object attribute class after the 60 episode initialisation setup, compared to a ground truth user distribution.

The threshold is defined as the minimum preference probability difference between semantic attributes for each object class. As an example, if the learned user profile for the three type attributes {*box, ball, key*} is $\{0.67, 0.28, 0.05\}$ respectively, we define the uncertainty threshold to be $0.28 - 0.05 = 0.23$. If a user gives an instruction to "pick up a blue object", in an environment with a blue box and a blue ball, the agent would compare the preference probability between these object types. Here the difference in preference probability is $0.67 - 0.28 = 0.39$ which greater than the threshold of 0.23, in this case it determines that user would prefer the "box" and composes the compositional value functions to act in the environment. If the composed task is incorrect, the agent will be able to ask a clarifying question to resolve the ambiguity and complete the task, updating the user profile accordingly.

To test the effectiveness of the user preference component defined in Section 3.3.2, we implement experimental settings for both skewed and flat user preference distributions and deter-

mine if our method generalises to various user profiles. A skewed user preference distribution is one where the user has a clear preference for a specific semantic attribute, whereas a flat distribution is one in which the users preferences are relatively equal across semantic attributes. The skewed and flat probability distributions (used as user preference ground truths for our experimentation) for each object attribute class are defined in Table 4.4.

Distribution Type	**Colour Preference Distribution** ['red', 'green', 'blue', 'yellow', 'grey', 'purple']	**Type Preference Distribution** ['box', 'key', 'ball']
Flat	$[0.19, 0.15, 0.16, 0.18, 0.16, 0.16]$	$[0.37, 0.33, 0.30]$
Skewed	$[0.03, 0.02, 0.84, 0.01, 0.06, 0.04]$	$[0.04, 0.05, 0.91]$

Table 4.4: The skewed and flat probability distributions defined for each object attribute class

Accuracy The accuracy results for our user preference tracking across each experiment setup are shown in Table 4.5. In line with the previous experiments, ambiguity is introduced across each of the object attribute classes separately and averaged over 100 episodes, for 10 runs.

We determine accuracy when an agent is only able to use the learned user profile to infer the correct task (UP), choosing the semantic object attribute with the highest probability to resolve any ambiguity, and compare this to using the user profile with the ability to ask a clarifying question if uncertain (UP+Q). The accuracy, seen in Table 4.5 is obtained for all four combinations of the user preference distributions, given that there are at least two ambiguous distractor objects in the environment.

Colour Distribution Setup	**Type Distribution Setup**	**Accuracy (UP)**	**Accuracy (UP+Q)**
Flat	Flat	87.75% $\pm$ 5.71%	**94.24**% $\pm$ 3.88%
Skewed	Flat	93.43% $\pm$ 2.13%	**95.54**% $\pm$ 1.91%
Flat	Skewed	90.32% $\pm$ 6.09%	**93.92**% $\pm$ 3.12%
Skewed	Skewed	95.09% $\pm$2.21%	**95.68**% $\pm$ 2.23%

Table 4.5: Two-Attribute Environment: The accuracy, when an ambiguous command is given by the user with two ambiguous distractor objects in the environment sharing at least one attribute with the goal object. The ambiguity is introduced across each of the object attributes separately. The accuracy for each case is combined and averaged over 100 episodes, with the standard deviations over 10 runs.

The results show that the user preference distribution (skewed or flat) has a marked affect on the accuracy, especially in the (UP) setting where the agent does not have an option to ask a clarifying question. The ability to ask a clarifying question in conjunction with tracking user preference (UP+Q) results in an average 3.07% increase in accuracy across the four experiments, highlighting the effectiveness of including the option to ask a clarifying question. We also see that the variances in (UP+Q) setting are much lower than that seen in the (UP) settings, demonstrating that it is far more reliable on average. Noticeably when the colour preference distribution is flat we see the largest performance decreases. The colour class has six semantic attributes that the agent has to track and resolve for, therefore when the probability difference between semantic attributes is very low the agent is not always able to resolve the ambiguity between preferences in the given time-steps.

Going forward for comparison purposes to other methods we use the results for the optimal case of having a user with skewed preference distributions for both object attribute classes. The optimal (UP+Q) setting shows only a marginal decrease of 1.75% compared to the unambiguous baseline and 1.52% compared to asking a question upfront (Q), for the environment setup with

two ambiguous distractor objects. These results show that the (UP+Q) setting is able to perform similarly to the unambiguous baseline setting as well as the relatively easier (Q) setting where the ambiguity is resolved upfront.

Environment Interactions We obtain the average number of interactions by the agent in an environment containing at least two ambiguous distractors of a class, in the skewed preference distribution setting, seen in Figure 4.11. Compared to the results for the easier (Q) setting, for the two-attribute environment, the average number of interactions when using the (UP+Q) algorithm is 6.62 when ambiguity is introduced by distractors sharing the same Colour, compared to the same Type attribute at 7.37. This is a 9.05% increase on average. The increase in interaction with the environment is expected, compared to the less complex (Q) setting, as a user's choice for an episode might not correlate directly to the learned preferences, in which case the agent will try collect the incorrect object before asking a question to clarify. However, as seen above in Table 4.5, the accuracy is only marginally affected despite the increase in interactions. We also note that the largest difference is seen when distractors share the same Type attribute, as mentioned previously this is due to the number of colour attributes needed to be resolved to complete the task for this setting.

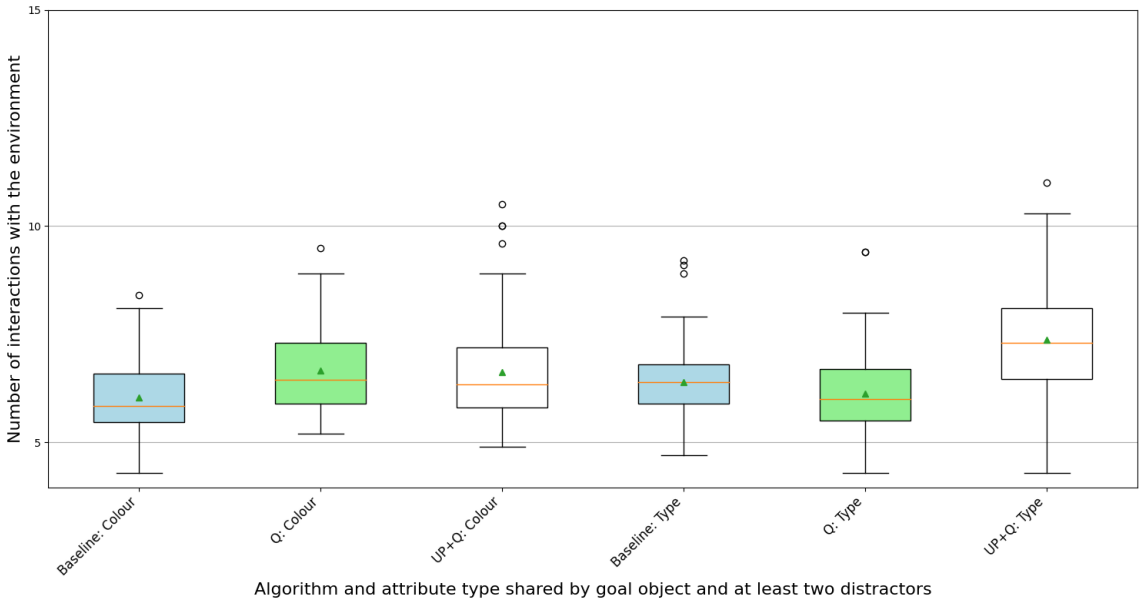


Figure 4.11: Two-attribute Environment: Average episode length for each object attribute class across various algorithms in an environment with two ambiguous distractors, averaged over 100 episodes for 10 runs. The baseline results are highlighted in blue and represent the results seen in the simpler unambiguous case, the results seen in the (Q) setting are highlighted in green.

User Interactions In Figure 4.12 we compare the average number of interactions by the agent with the user for the (UP+Q) setting, to the unambiguous baseline and (Q) settings, when there are at least two ambiguous distractors present in the environment. As expected asking a clarifying question upfront (Q) results the highest number of interactions for the user, in comparison to the (UP+Q) setting (with ~ 0.12 questions per episode on average). We can see from these results that, having learned the user profile the agent can successfully act in the environment without having to interact with the user in the majority of cases. This also demonstrates the noticeable improvement in user experience for the (UP+Q) setting, where the agent does not have to ask the user clarifying questions, but is still able to resolve ambiguity and successfully complete the tasks with a high accuracy, as discussed above.

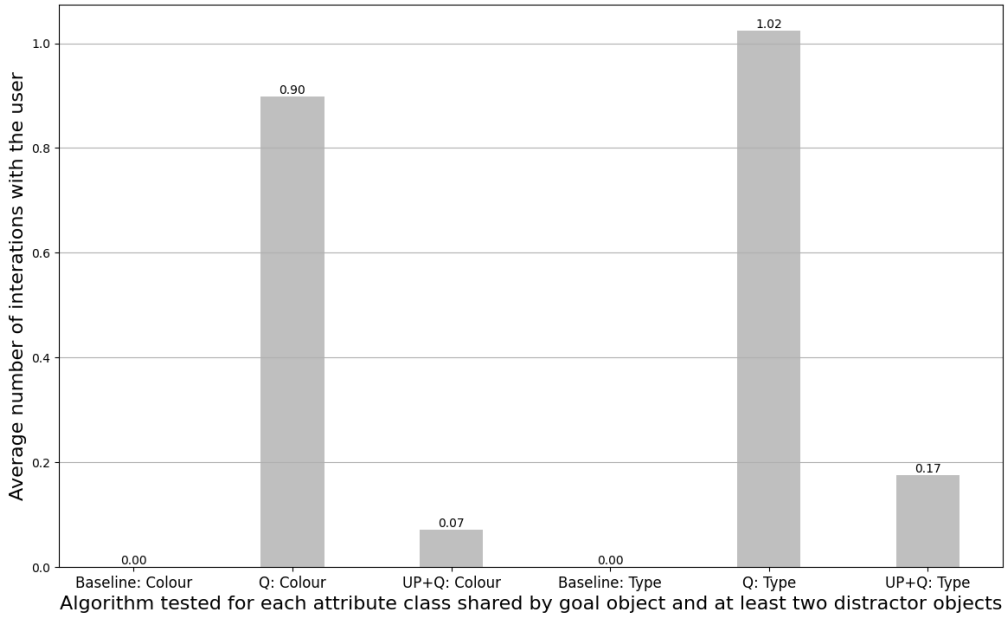


Figure 4.12: Two-attribute Environment: Average number of user interactions for each object attribute class across various algorithms in an environment with two ambiguous distractors, averaged over 100 episodes for 10 runs.

Scalability To determine how well the (UP+Q) setting scales, we implement it in the three-attribute environment and obtain the accuracy (Table 4.6), environment interactions and user interactions (Figure 4.13).

No. of Ambiguous Distractors	**Accuracy (UP+Q)**	Δ Baseline
2	95.05% $\pm$ 1.86%	-2.03%

Table 4.6: Three-Attribute Environment: The accuracy, when an ambiguous command is given by the user with two ambiguous distractors. The ambiguity is introduced across each of the object attributes separately, combined and averaged over 100 episodes. With the standard deviations over 10 runs.

The accuracy for the (UP+Q) experimental setting is 2.03% less accurate than the easier unambiguous baseline result and 1.55% less accurate then the (Q) results, while the average number of interactions, is $6.79, 7.00$ and 6.67 when ambiguity is introduced in the *Colour*, *Type* and *Material* class respectively, which is a minimal 2.0% increase on average. These results are in line with what we see in the two-attribute environment metrics.

As with the two-attribute environment, the largest difference is seen in the user interactions, as the agent has to ask < 2 questions per episode on average in the (UP+Q) setting, which results in an average of 0.42 less questions than is seen in the (Q) setting. Overall we note that the (UP+Q) setting is able to scale sufficiently when there is added complexity in the environment. This result again demonstrates the noticeable improvement in user experience for the (UP+Q) setting compared to the (Q) setting, while still achieving a high accuracy.

In summary we see that the (UP+Q) setting is able to decrease the number of interactions to almost ~ 0 in the two-attribute case while maintaining an overall accuracy of 95.68%. It is noted that the number of environment interactions increase, compared to the easier (Q) setting, but this is outweighed by the increase in usability of the system, as the agent does not have ask the user a question for every interaction. From these results we determine that the proposed

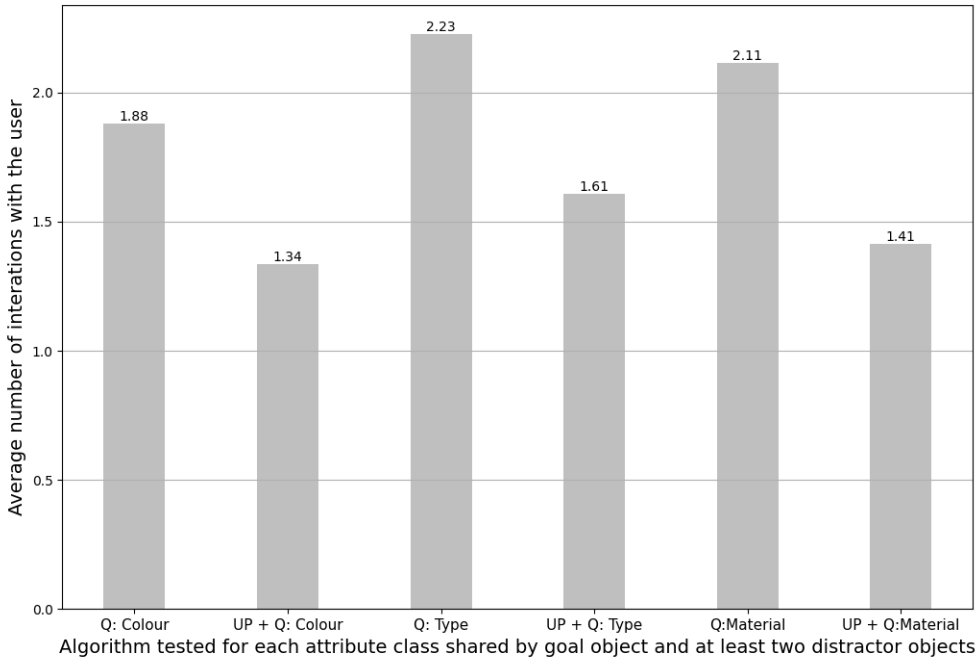


Figure 4.13: Three-Attribute Environment: Average number of user interactions for each object attribute class across various algorithms in an environment with two ambiguous distractors, averaged over 100 episodes for 10 runs.

framework of adaptively inferring the complete task instruction, should ambiguity exist, by using clarifying questions in conjunction with dynamic user preference tracking, adequately addresses the problem of ambiguity in HRI allowing for a better user experience, while suffering minimal losses in accuracy.

4.3 Natural Language Commands

As a final step we implement the dynamic translation and note any degradation in performance accuracy resulting from the introduction of the NMT model on the complete system. The system makes use of the pre-trained T5 [Raffel *et al.* 2019] transformer model setup described in Cohen *et al.* [2021] and Section 2.1.2, to translate referring expressions (e.g. "pick up the ball") to the logical expression *(pickup_ball)*, as shown in Figure 3.1b. The logical Boolean expression, defined by the output token, represents the learned value function ($\bar{Q}^*_B$). In line with Cohen *et al.* [2021] the T5 seq2seq transformer model is fine-tuned using RL, with a reward of +1.0 and -1.0 for the syntactically correct or incorrect Boolean output expression respectively. During training the sampled Boolean expressions are compared to the ground truth Boolean expressions for those tasks [Cohen *et al.* 2021], and after 100 training steps the performance of the agent is evaluated using returns from 100 inference iterations. A translation task is considered solved when an inference result of 95% is achieved. We modify the input referring expressions to include one word utterance inputs ("ball"), to mimic a user answer to a question posed by the agent. During inference a Boolean algebra expression parser is used to validate the expression and the compositional value function is then obtained and used by the agent to act in the environment.

The accuracy results are provided in Table 4.7 for the (UP+Q) setting with skewed user preference distributions for comparison to previous results, in the two-attribute environment, with at least two ambiguous distractors.

No. of Ambiguous Distractors	Accuracy (UP+Q)	$\triangle$ (UP+Q) without NMT
2	94.41% $\pm$2.45 %	-1.27%

Table 4.7: Two-Attribute Environment: The accuracy when implementing the fine-tuned T5 NMT model, when an ambiguous command is given by the user with two ambiguous distractors in the environment. The ambiguity is introduced across each of the object attributes separately, combined and averaged over 100 episodes. With the standard deviations over 10 runs.

There is a 1.27% decrease in accuracy when adding the fine-tuned NMT model to the complete system. This decrease is due to inference inaccuracies resulting from incorrect output tokens that do not correspond to a compositional value function, which the agent needs in order to act in the environment and complete a task. The NMT outputs 98 incorrect tokens in the 1000 episodes (i.e. 100 episodes for 10 runs), which is an error rate of 9.8%. This is hypothesied to be as a result of the stopping criteria used during fine-tuning and it is noticed that the accuracy needs to be increased for this case in future work. Despite this, the high overall system accuracy of 94.41% with all the components included indicates the viability of the system of the use case of addressing ambiguity in HRI resulting from incomplete referring expressions.

4.4 Summary

This chapter describes the experimentation implemented to test the components of our methodology and address uncertainty arising from ambiguous referring expressions. The components include determining the performance when asking a clarifying question upfront (Q) to resolve ambiguity, followed by the implementation of user preference tracking over the object semantic properties with the option to ask a clarifying question when the agent is uncertain (UP+Q). The different components were compared to each other, and the unambiguous baseline, across three specific criteria 1) the accuracy of the algorithm, 2) the number of interactions by the agent with the environment and 3) the number of interactions with the user. We empirically determine that user preference tracking with the option to ask a clarifying question (UP+Q) when uncertain (for skewed user preference distributions) results in the lowest number of interactions with the user (seen in Figure 4.12) while still achieving a high accuracy (seen in Table 4.5), therefore providing a better user experience.

Chapter 5

Conclusion

Social robotics, integrating with the lives of humans, require natural human collaboration and communication abilities. To facilitate a natural verbal interaction, social robots need to understand the information/instruction from the user and perform tasks accordingly. One key challenge is that natural language can be a source of ambiguity, especially when there is also ambiguity in the environment (i.e. similar objects to be retrieved). This research proposes a framework that leverages the compositionality of learned object-attribute base-tasks in conjunction with user preference and clarifying questions for adaptive task inference, to address ambiguity in HRI and perform tasks specified by a natural language instruction. The research consists of training and testing compositional agents that follow language commands. We define a method for adaptively inferring the complete task instruction, by leveraging learned base-tasks, when ambiguity is introduced. The method includes the implementation of a question and answering algorithm to clarify ambiguity; and the application of dynamic user preferences to prevent unnecessary repeated clarifying interactions. We extensively test all components of our system and determine that our presented method provides a viable solution for addressing the problem of ambiguity in HRI, by decreasing the number of interactions with the user while maintaining a high level of accuracy.

5.1 Future Work

The various components of the system allow for multiple possible avenues of future work to be presented. For the training of compositional agents, the work by Tasse *et al.* [2020] defines the conjunction, disjunction and negation, however we only focus on the conjunction of tasks for this research. Including the possibility to perform disjunction and negation would result in a exponential increase in the number of novel tasks the agent is able to solve. We also note that our work assumes the described objects are in the robot's viewpoint in a fixed scene, and does not consider objects outside of the scene, which presents another possible avenue for further exploration.

For the user preference profiling we implement an initialisation step, however it is noted that this could also be performed using online learning [Gan *et al.* 2019], investigating whether the user experience would suffer significantly from poor initial recommendations in online learning would be an interesting investigation for future work. Similarly inference from inter-user preferences has not been considered in the current experimentation and user profile tracking is done on a per person basis, however the inclusion of this could allow for a better user experience.

This research explicitly expands on previous research by Cohen *et al.* [2021], and the T5 transformer architecture [Raffel *et al.* 2019] was kept the same. This aligned implementation, in the same environment, provides a comparable evaluation baseline which is used to determine how well our system performs. However the examination of different NMT transformer architectures is noted as possible future work.

Additionally exciting future directions include extending the natural language capabilities of the agent to include a multilingual setting, where a user is able to instruct the robot in any language; and the extraction of "common sense" information [Zhao *et al.* 2024] from a Large Language Model based on the context of the language command i.e. picking up a porcelain cup rather than a glass based on the context of the request.

References

[Ackley *et al.* 1985] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985.

[Akalin and Loutfi 2021] Neziha Akalin and Amy Loutfi. Reinforcement learning approaches in social robotics. *Sensors*, 21(4), 2021.

[Ba *et al.* 2016] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*, 2016.

[Bahdanau *et al.* 2014] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[Bamdale *et al.* 2019] Rishikesh Bamdale, Shreejeet Sahay, and Varsha Khandekar. Natural human robot interaction using artificial intelligence: A survey. In *2019 9th Annual Information Technology, Electromechanical Engineering and Microelectronics Conference (IEMECON)*, pages 297–302, 2019.

[Bellman 1957] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[Bertsekas and Tsitsiklis 1991] D. Bertsekas and J. Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, 1991.

[Chevalier-Boisvert *et al.* 2018] Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. *BabyAI: A Platform to Study the Sample Efficiency of Grounded Language Learning*, 2018.

[Cohen *et al.* 2021] Vanya Cohen, Geraud Nangue Tasse, Nakul Gopalan, Steven James, Matthew Craig Gombolay, and Benjamin Rosman. Learning to follow language instructions with compositional policies. *ArXiv*, abs/2110.04647, 2021.

[Doğan *et al.* 2022] Fethiye Irmak Doğan, Ilaria Torre, and Iolanda Leite. Asking follow-up clarifications to resolve ambiguities in human-robot conversation. In *Proceedings of the 2022 ACM/IEEE International Conference on Human-Robot Interaction*, HRI '22, pages 461–469. IEEE Press, 2022.

[Doğan 2021] Fethiye Irmak Doğan. Social robots that understand natural language instructions and resolve ambiguities. In *Robotics: Science and Systems (RSS) Pioneers Workshop 2021*, 2021.

[Doğan 2023] F.I. Doğan. *Robots That Understand Natural Language Instructions and Resolve Ambiguities*. TRITA-EECS-AVL. KTH Royal Institute of Technology, 2023.

[Fan *et al.* 2020] Angela Fan, Shruti Bhosale, Holger Schwenk, Zhiyi Ma, Ahmed El-Kishky, Siddharth Goyal, Mandeep Baines, Onur Celebi, Guillaume Wenzek, Vishrav Chaudhary, Naman Goyal, Tom Birch, Vitaliy Liptchinsky, Sergey Edunov, Edouard Grave, Michael

Auli, and Armand Joulin. Beyond english-centric multilingual machine translation. *CoRR*, abs/2010.11125, 2020.

[Gan *et al.* 2019] Chao Gan, Jing Yang, Ruida Zhou, and Cong Shen. Online learning with diverse user preferences. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 2539–2543, 2019.

[Hatori *et al.* 2018] Jun Hatori, Yuta Kikuchi, Sosuke Kobayashi, Kuniyuki Takahashi, Yuta Tsuboi, Yuya Unno, Wilson Ko, and Jethro Tan. *Interactively Picking Real-World Objects with Unconstrained Spoken Language Instructions*, 2018.

[He *et al.* 2015] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[Hellou *et al.* 2021] Mehdi Hellou, Norina Gasteiger, Jong Yoon Lim, Minsu Jang, and Ho Seok Ahn. Personalization and localization in human-robot interaction: A review of technical methods. *Robotics*, 10(4), 2021.

[Jaksch *et al.* 2010] Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11(51):1563–1600, 2010.

[Lee *et al.* 2012a] Min Kyung Lee, Jodi Forlizzi, Sara Kiesler, Paul Rybski, John Antanitis, and Sarun Savetsila. Personalization in hri: A longitudinal field experiment. pages 319–326, 03 2012.

[Lee *et al.* 2012b] Min Kyung Lee, Jodi Forlizzi, Sara Kiesler, Paul Rybski, John Antanitis, and Sarun Savetsila. Personalization in hri: A longitudinal field experiment. pages 319–326, 03 2012.

[Mason and Lopes 2011] Martin Mason and Manuel Lopes. Robot self-initiative and personalization by learning through repeated interactions. In *2011 6th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 433–440, 2011.

[Matuszek 2018] Cynthia Matuszek. Grounded language learning: Where robotics and nlp meet. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 5687–5691. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[Mnih *et al.* 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[Muthugala and Jayasekara 2018] Viraj Muthugala and Buddhika Jayasekara. A review of service robots coping with uncertain information in natural language instructions. *IEEE Access*, PP:1–1, 02 2018.

[Parisi] Dr. Parisi. *Lifelong Learning and Long-Term Human-Robot Interaction*.

[Puterman 2014] Martin L Puterman. *Markov Decision Processes.: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[Raffel *et al.* 2019] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683, 2019.

[Selvaraju *et al.* 2017] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 618–626, 2017.

[Shridhar *et al.* 2020] Mohit Shridhar, Dixant Mittal, and David Hsu. Ingress: Interactive visual grounding of referring expressions. *The International Journal of Robotics Research*, 39:027836491989713, 01 2020.

[Sutton *et al.* 1998] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[Tasse *et al.* 2020] Geraud Nangue Tasse, Steven James, and Benjamin Rosman. *A Boolean Task Algebra for Reinforcement Learning*, 2020.

[Vaswani *et al.* 2017a] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*, 2017.

[Vaswani *et al.* 2017b] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[Whitney *et al.* 2016] David Whitney, Miles Eldon, John Oberlin, and Stefanie Tellex. Interpreting multimodal referring expressions in real time. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3331–3338, 2016.

[Whitney *et al.* 2017] David Whitney, Eric Rosen, James MacGlashan, Lawson L. S. Wong, and Stefanie Tellex. Reducing errors in object-fetching interactions through social feedback. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1006–1013, 2017.

[Yang *et al.* 2022] Yang Yang, Xibai Lou, and Changhyun Choi. Interactive robotic grasping with attribute-guided disambiguation. *arXiv preprint arXiv:2203.08037*, 2022.

[Young *et al.* 2010] Steve Young, Milica Gašić, Simon Keizer, François Mairesse, Jost Schatzmann, Blaise Thomson, and Kai Yu. The hidden information state model: A practical framework for pomdp-based spoken dialogue management. *Computer Speech and Language*, 24(2):150–174, 2010.

[Zhang *et al.* 2023] Ceng Zhang, Junxin Chen, Jiatong Li, Yanhong Peng, and Zebing Mao. Large language models for human–robot interaction: A review. *Biomimetic Intelligence and Robotics*, 3(4):100131, 2023.

[Zhao *et al.* 2024] Zirui Zhao, Wee Sun Lee, and David Hsu. Large language models as commonsense knowledge for large-scale task planning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2024. Curran Associates Inc.

Appendix

Appendix A

The architecure of the DQNs used to learn the base taks value functions. The CNN follows the same structure used [Mnih *et al.* 2015].

Convolutional Layers	
Layer 1	3 input channels, 32 output channels, a kernel size of 8 and a stride of 4
Layer 2	32 input channels, 64 output channels, a kernel size of 4 and a stride of 2
Layer 3	64 input channels, 64 output channels, a kernel size of 3 and a stride of 1
Two fully-connected linear layers	
Layer 1	Input size 3136 and output size 512, makes use of a ReLU activation function
Layer 2	Input size 512 and output size 7 with no activation function

Table 5.1: DQN Architecture: As in [Cohen *et al.* 2021] we used the ADAM optimiser with batch size 256 and a learning rate of 10^{-3}. The target Q-network was updated every 1000 steps, and we used greedy exploration, annealing from 1.0 to 0.1 over 1000000 time steps.