

Towards More Accessible Human-AI Interactions in Sequential Decision-making

Tasks

by

Utkarsh Soni

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved April 2024
by the Graduate Supervisory Committee:

Subbarao Kambhampati, Chair
Chitta Baral
Chris Bryan
Huan Liu

ARIZONA STATE UNIVERSITY

May 2024

ABSTRACT

In today’s world, artificial intelligence (AI) is increasingly becoming a part of our daily lives. For this integration to be successful, it’s essential that AI systems can effectively interact with humans. This means making the AI system’s behavior more understandable to users and allowing users to customize the system’s behavior to match their preferences. However, there are significant challenges associated with achieving this goal.

One major challenge is that modern AI systems, which have shown great success, often make decisions based on learned representations. These representations, often acquired through deep learning techniques, are typically inscrutable to the users inhibiting explainability and customizability of the system. Additionally, since each user may have unique preferences and expertise, the interaction process must be tailored to each individual.

This thesis addresses these challenges that arise in human-AI interaction scenarios, especially in cases where the AI system is tasked with solving sequential decision-making problems. This is achieved by introducing a framework that uses a symbolic interface to facilitate communication between humans and AI agents. This shared vocabulary acts as a bridge, enabling the AI agent to provide explanations in terms that are easy for humans to understand and allowing users to express their preferences using this common language. To address the need for personalization, the framework provides mechanisms that allow users to expand this shared vocabulary, enabling them to express their unique preferences effectively. Moreover, the AI systems are designed to take into account the user’s background knowledge when generating explanations tailored to their specific needs.

ACKNOWLEDGMENTS

My PhD journey has been marked by challenges, personal growth, and invaluable learning experiences. As I reflect on this achievement, I am profoundly grateful to those who have supported me every step of the way.

First and foremost, I extend my heartfelt gratitude to my advisor, Dr. Rao. His unwavering guidance, support, and wisdom have been instrumental in shaping both my academic journey and professional growth. His commitment to bringing in new ideas and his invaluable advice to explore new frontiers rather than perfecting existing ones have left a mark on my approach to professional life. Moreover, his dedication to his family and his humility is something that I truly admire. I am truly fortunate to have had the opportunity to work alongside such a remarkable mentor. I know I never said it out loud, but in my mind he has always been Superrao who I can always rely to swoop in and save the day.

I am also indebted to my lab mates for their collaboration, and insightful contributions to my work. Each of you has played a pivotal role in enriching my research experience. I want to express my special appreciation to Sarath, whose insights and balanced perspective have guided me through various challenges. Sriram's brilliance and ability to engage in meaningful conversations have significantly enhanced how I communicate. Yantian's relentless dedication and resilience have been a constant source of inspiration, reminding me of the importance of never giving up. And to Lin, Mudit, Siddhant, and Zahra, your passion and dedication have motivated me to strive for excellence.

Last, but certainly not least, I am deeply grateful to my family, my girlfriend Aditi, and my friends Yatharth and Ishita. Your unwavering love, encouragement, and understanding have been my pillars of strength throughout this journey and I would always be indebted for all the sacrifices you have made to support me.

TABLE OF CONTENTS

	Page
LIST OF TABLES	vi
LIST OF FIGURES	vii
CHAPTER	
1 INTRODUCTION	1
1.1 Defense Outline	3
2 BACKGROUND	5
2.1 Generating Explanations as Model Reconciliation	5
2.2 Visual & Symbolic Explanations for Making Inscrutable Agents Interpretable	7
2.3 Expressing User’s Preferences to Agents with Inscrutable Task Rep- resentation	9
3 GENERATING EXPLANATIONS FOR AGENT WITH INSCRUTABLE TASK REPRESENTATION	11
3.1 Problem Setting	13
3.2 Contrastive Explanations	15
3.2.1 Learning Concept Maps	16
3.2.2 Symbolic explanation using concepts	17
3.3 Identifying Explanations Through Sample-Based Trials	19
3.3.1 Identifying failing precondition	19
3.3.2 Identifying cost function	21
3.4 Quantifying Explanatory Confidence	22
3.4.1 Graphical model for computing confidence in explanations ..	23
3.4.2 Posterior probability of a concept being a precondition	24
3.4.3 Posterior probability for cost function abstraction	26

CHAPTER	Page
3.4.4	Modified algorithm accounting for confidence in explanations 27
3.5	Evaluation 27
3.5.1	Computational Experiments 30
3.5.2	User study 31
3.6	Chapter Summary 35
4	GENERATING PERSONALIZED EXPLANATIONS IN SEQUENTIAL DECISION MAKING PROBLEM 36
4.1	Illustrative Example 39
4.2	Background 40
4.2.1	Learning approximate mental model of the user to generate explanations 41
4.3	Explanation in the Presence of Multiple User Types 42
4.3.1	Learning the set of labeling models 42
4.3.2	Explaining Behavior Traces 45
4.3.3	Myopic Approximation: QMDP 50
4.3.4	Myopic Approximation: QHR 51
4.3.5	POMCP with d-step lookahead 52
4.4	Evaluation 52
4.4.1	Computational experiments 52
4.4.2	User study 57
4.5	Chapter Summary 61
5	VOCABULARY EXPANSION FOR PREFERENCE SPECIFICATION IN INSCRUTABLE TASK REPRESENTATION 62
5.1	Illustrative Example 67

CHAPTER	Page
5.2 Problem Setting	69
5.3 Assumptions for Heuristic Label Estimation	70
5.3.1 Causal relationship with a known concept	71
5.3.2 Concept proximity and rarity	72
5.3.3 Concept image locality	73
5.4 Adaptive Supervised Learning to Learn the Target Concept	73
5.4.1 Labeling scheme for fully supervised learning to train con- cept classifier	74
5.4.2 Labeling scheme for weak supervised learning	75
5.4.3 Using pretrained vision language models for feedback-efficient training	78
5.4.4 Adaptive supervised training.....	79
5.5 Using the Learned Concept During Training	82
5.6 Evaluation	84
5.6.1 Computational experiments.....	84
5.6.2 User study	91
5.7 Chapter Summary	97
6 CONCLUSION.....	98
REFERENCES	100
7 ASU IRB APPROVAL.....	105

LIST OF TABLES

Table		Page
3.1	Evaluation results for testing whether participants preferred symbolic explanations indicating model components	32
3.2	Evaluation results for testing whether participants preferred concept based explanations over saliency maps	33
4.1	Inaccurate model components for each possible user-type of the disaster resuce domain	38
4.2	Average regret for the POMDP solvers and baseline technique along with the interaction cost, $C(\mathcal{E})_{pc}$, incurred by an oracle agent for varying value of λ	56
4.3	Evaluation Measures for comparing <i>personalized</i> and <i>conformant</i> explanations	60
5.1	Comparing the cognitive load for concept query vs pairwise comparison query. Each row represents one of the dimension used in NASA TLX workload index. We adapted the questions to let participant choose which query was better for each dimension. The participants chose concept query to be better for all the dimensions, and the preference was found to be statistically significant in each case.	96

LIST OF FIGURES

Figure		Page
1.1	Overall architecture of our proposed AI system that exposes a symbolic vocabulary interface to the user thus enabling the AI agent to generate explanations in human interpretable vocabulary and letting the human convey their preference to the agent. In addition, the user can add new terms to the interface when the vocabulary is insufficient for their preference. Finally, the agent reasons about the current user's type & generates an explanation tailored to their type.....	3
3.1	The explanatory dialogue starts when the agent's plan incorporates jumping over the skull, prompting the user to propose specific alternative plans. The system then explains its preference for the original plan using model information in terms of propositional concepts that the user understands. These concepts themselves are grounded in the agent's task representation as classifiers.	12
3.2	Simplified probabilistic graphical models for confidence of learned (A) Preconditions and (B) Cost.	23
3.3	The plan and foils used in the evaluation for the game of Montezuma's revenge in two different levels. Plans are highlighted in blue and foils in red. The concepts corresponding to missing preconditions are <code>not_on_rope</code> , <code>not_on_left_ledge</code> , <code>not_skull_on_left</code> and <code>is_clear_down_of_crab</code> for failure points A, B, C and D respectively (note all concepts were originally defined as positives and we formed the <code>not_</code> concepts by negating the output of the classifier).	28

3.4	The plan and foils used in the evaluation for the two variants of the Sokoban game (sokoban-switch on the left and sokoban-cell on the right). Plans are highlighted in blue and foils in red. The concepts corresponding to the cost function, for the Sokoban cost variants are switch_on and on_pink_cell that results in push action costing 10 (instead of 1)	29
3.5	The average probability assigned to the correct model component by the search algorithms, calculated over ten random search episodes with std-deviation.	30
3.6	The saliency map based explanation shown to users to test hypothesis H2.	34
4.1	An instance of disaster-rescue domain along with a robot's execution trace with transitions labeled with their corresponding rewards	38
4.2	Web interface for human-robot interaction during the study, where participants provide explicability labels for each transition while receiving plan prefixes and explanations.	58
5.1	Overview of <i>PRESCA</i> . (1) The user specifies their preference in terms of some symbolic concept. If the concept is not present in the symbolic interface, then the user provides its causal relationship to some known concept. (2) <i>PRESCA</i> then generates likely positive examples and negative examples of the concept and queries their label to the user. (3) After getting the labels, <i>PRESCA</i> learns a classifier for the target concept and (4) incorporates user's preference in agent's training. (5) Finally, the concept is added to the interface.	66

5.2	(a) Instance of the Minecraft environment with two possible plans marked with arrows. The user prefers that the agent avoid going into the storage area (indicated by the red arrows) (b) the causal model of the Minecraft environment.	67
5.3	Annotated patch corresponding to the concept in storage area.....	73
5.4	An example state image of the Minecraft domain with all objects annotated	85
5.5	Example patches annotated with red bounding box for the concept (a) has broken ladder and (b) has ladder	85
5.6	F1 score of the trained classifier for the concept <code>in_storage_area</code> against the total feedback for the case where number of intermediate concepts = 0	87
5.7	F1 score of the trained classifier for the concept <code>in_storage_area</code> against the total feedback for the case where number of intermediate concepts = 1	88
5.8	F1 score of the trained classifier for the concept <code>in_storage_area</code> against the total feedback for the case where number of intermediate concepts = 2	88
5.9	Total feedback vs average reward per map	89
5.10	Domain used for the user study. The robot's task is to reach the goal (cell with yellow edges), while avoiding going over the carpet area (cell marked as green square). The domain has 5 possible maps with varying locations of the goal and the carpet.....	92
5.11	An example of a <i>concept query</i> for the concept <code>robot_on_carpet</code>	93

5.12	An example of a <i>pairwise comparison</i> query. When clicked, the clip will play, showing the robot’s actions in the game. Participants are tasked with selecting the clip where the robot’s performance is better. They may also choose the “tie” option to indicate similar performance or select the “can’t say” option if they are unable to make the comparison.	94
7.1	IRB approval page-1	105
7.2	IRB approval page-2	106

Chapter 1

INTRODUCTION

In recent years, significant advancements in AI have generated increasing interest in integrating AI systems into our daily lives. To successfully deploy such systems, it's crucial that these AI systems exhibit behavior that is comprehensible to humans and aligns with their preferences. A crucial step toward achieving this is establishing effective human-AI interaction. This means giving AI agents the ability to explain their decisions to humans when confusion arises about the system's behavior and ensuring that humans can instruct the agent to follow their preferences.

While AI systems have made remarkable progress in performing complex tasks like autonomous driving, robotic manipulation, assembly, healthcare automation, financial advising, and surveillance, there is a notable gap when it comes to ensuring that these AI agent's behavior is comprehensible and adaptable by the users who ultimately interact with them. This challenge arises because these systems heavily depend on internal task representations that have been learned using deep learning methods. Unfortunately, these learned representations are opaque for the end users, creating a substantial barrier in establishing effective communication between humans and AI systems.

Another key challenge in human-AI interaction is personalization. The AI system needs to adapt to individual users' unique preferences. Thus, communication framework should be versatile enough to adjust to each user. However, this can get complicated due to the inscrutability of the agent's model. Additionally, considering that users may have varying background knowledge and expertise, AI systems should customize explanations to match the user's understanding of the task without

overwhelming them with too much detail.

This thesis focuses on scenarios where AI systems are designed to solve sequential decision-making problems. In this context, our primary goal is to address the challenge of dealing with inscrutable task models and enabling personalized human-AI interaction. Although our primary focus is on sequential decision-making, most of the principles and techniques we study can be applied to human-AI interaction settings in general.

To tackle the challenge of communicating with AI agents whose task models are incomprehensible to users, current AI techniques often depend on using shared medium, such as images of the environment or videos of the agent’s behavior, to facilitate the interaction. This may involve using saliency regions in images to interpret the agent’s decisions (Greydanus *et al.* (2018)) or using Preference based reinforcement learning (PbRL) that employ pairwise behavior comparison to learn user’s preferences (Christiano *et al.* (2017)). However, such systems may not scale well in complex sequential decision-making scenarios, as this form of information exchange can place a high cognitive load on humans and be a weak training signal for the agent.

In this thesis, we advocate that human-AI communication be conducted through a symbolic interface consisting of terms that humans understand and that are grounded in the agent’s task representation. We design systems capable of leveraging this shared vocabulary for communication. The agent generates explanations using this vocabulary, while the user uses it to express their preferences.

Furthermore, we make sure that the interaction is personalized for the user. This is achieved by allowing the human to add new terms to the vocabulary to convey their unique preferences. Additionally, our agents are equipped with the capability to consider the user’s background knowledge of the task and tailor the explanations they provide to suit the specific user. This builds upon the existing work in explainable

AI for sequential decision problems (Sreedharan *et al.* (2022)).

The overall architecture of our proposed system for creating advisable and explainable AI agents is depicted in Figure 1.1.

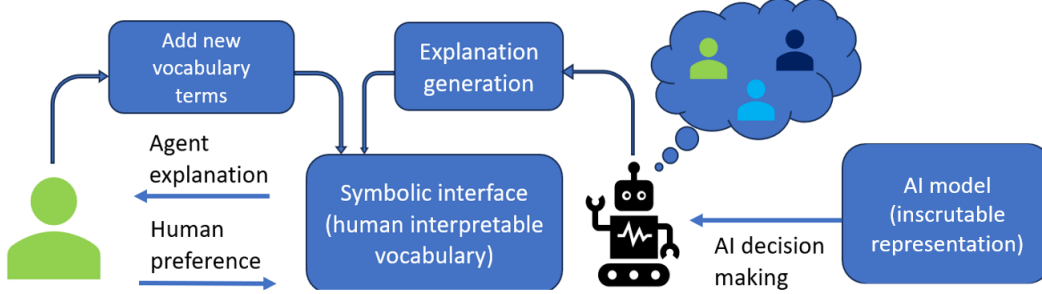


Figure 1.1: Overall architecture of our proposed AI system that exposes a symbolic vocabulary interface to the user thus enabling the AI agent to generate explanations in human interpretable vocabulary and letting the human convey their preference to the agent. In addition, the user can add new terms to the interface when the vocabulary is insufficient for their preference. Finally, the agent reasons about the current user’s type & generates an explanation tailored to their type.

1.1 Defense Outline

The following sections of this defense are structured as follows:

Chapter 2 : In this chapter, we provide an overview of current research on explainable and advisable AI agents for sequential decision-making problems. These agents generate explanations while considering the human’s model of the task. We note the limitations of these techniques in terms of producing personalized explanations. Following that, we critically assess prior research in explainable and advisable AI systems, with a particular emphasis on scenarios where the agent’s task representation is non-interpretable.

Chapter 3 : In this chapter, we explore the setting where the agent generates explanations in terms of a pre-established symbolic vocabulary. We demonstrate how the agent can learn an approximate symbolic representation of the task to generate such explanations. We also present the results of human subject studies that measure the preference for symbolic explanations and their impact on users’ understanding of the agent’s task model.

Chapter 4 : This chapter introduces our technique for enabling AI agents to personalize their explanations for users. We illustrate how the agent can identify user types relevant to the task it is solving and present a methodology for inferring the current user’s type and tailoring the agent’s explanation accordingly.

Chapter 5 : Here, we discuss our work that focuses on the challenge of specifying preferences using the symbolic interface. We particularly address situations where the existing vocabulary is insufficient and must be expanded for preference specification. We present a technique for introducing new vocabulary terms while requiring minimal feedback from the user. We perform extensive evaluation of this technique, including comparison with PbRL that employs non symbolic feedback from the user. We additionally perform a user study to compare the cognitive load of our method vs pairwise comparison queries opted by PbRL.

Chapter 6 : This chapter sets the conclusion of this work along with some suggested future directions.

Chapter 2

BACKGROUND

In this chapter, we begin by explaining the fundamental *model reconciliation* framework. This framework enables an agent to produce explanations for its actions by taking into account the human’s understanding of the task. We’ll also highlight the current limitations of this framework, which we aim to address in this thesis. The remainder of the chapter is then divided into two sections. The first section discusses how existing explainable AI techniques have dealt with the challenge of generating explanations for AI systems that use task representations that are non-interpretable to the user. The subsequent section outlines the techniques employed by these systems to allow users to provide guidance or instructions to the agent.

2.1 Generating Explanations as Model Reconciliation

In any interaction between humans and AI, it’s crucial to consider the user’s model of the task. Let’s imagine a situation where an AI agent creates an optimal plan to achieve its goal based on its own model of the task. In this scenario, there’s a human involved who’s responsible for evaluating the plan. Now, if the human’s model of the task differs from the system’s model, there’s a possibility that the human won’t agree with the quality of the plan. In such cases, it becomes important for the AI agent to provide an explanation for its plan that addresses this model difference. This explanation takes the form of model updates that aim to bring the human’s mental model closer to its own model such that the plan is optimal in the human’s updated mental model.

The concept of offering explanations through model updates was initially intro-

duced in the research by Chakraborti *et al.* (2017), where they called this process as *model reconciliation*. In their study, they operated under the assumption that both the system’s model and the human’s mental model, which was used to evaluate the plan, were known. The model updates were found by doing a systematic search in the space of models. The objective was to minimize the number of necessary model updates until a model was found that would evaluate the agent’s plan to be optimal.

This framework for generating explanations based on the user’s mental model is particularly relevant to this thesis. In the original work by Chakraborti *et al.* (2017), the human’s model was assumed to be known. Subsequent research, such as that by Sreedharan *et al.* (2018a) and Sreedharan *et al.* (2018b), addressed scenarios where uncertainty existed regarding the human’s model, and the AI agent attempted to customize explanations based on its estimation of the user’s model.

However, these works have certain limitations. In the case of Sreedharan *et al.* (2018a), the system requires prior knowledge of the possible mental models of users, and these models must be represented in a specific declarative form. Additionally, the user-adaptive explanation process in this work requires that the agent should have the ability to ask specific queries to the user to assess their knowledge about the task. Likewise, Sreedharan *et al.* (2018b) makes similar assumptions, requiring prior knowledge of the possible mental models of users. Furthermore, it restricts the user’s model to be an exact state abstraction of the system’s model.

In this thesis, we take a different approach by relaxing the assumptions concerning any prior knowledge of the potential mental models of users. Instead, we introduce data-centric techniques that allow us to learn these models. Our primary focus is on learning models for various user types with whom the agent may interact. Subsequently, the agent can reason about the current user’s type and provide a tailored explanation based on the learned model.

Another limitation of the original *model reconciliation* approach and its extensions is their assumption that the system’s task model is symbolic and uses terms that are understandable to the user, which makes the model space search feasible. However, this assumption doesn’t always hold, particularly when the agent employs deep learning-based function approximators to learn the task dynamics.

In this thesis, we establish a framework that specifically addresses this situation, where the agent’s model is inscrutable to the user. In the next section, we will explore previous research that has tackled the challenge of generating explanations in such scenarios.

2.2 Visual & Symbolic Explanations for Making Inscrutable Agents Interpretable

As previously mentioned, there are situations in human-AI interactions where the agent employs a task representation that is not understandable to the user. In sequential-decision making scenarios, these agents are typically trained using deep reinforcement learning (RL) techniques applied on image-based observations of the state. Currently, the go-to method for explaining how such agents make their decisions is through the use of saliency maps (refer to Zahavy *et al.* (2016), Greydanus *et al.* (2018), Iyer *et al.* (2018), Puri *et al.* (2019)). A saliency map is essentially a heat-map that illustrates which pixels the agent focused on when making its decision. However, unlike explanations in natural language, visual explanations like saliency maps can be mentally taxing for humans to interpret. Furthermore, as demonstrated in studies such as Atrey *et al.* (2019) and Kim *et al.* (2018), saliency maps can often be perplexing and result in an inaccurate understanding of the agent’s decision-making process.

Given the challenges linked with visual explanations, such as saliency maps, there’s a growing interest in exploring alternative methods for generating explanations. Sym-

bolic explanations, showcased in works like Kim *et al.* (2018), Ghorbani *et al.* (2019), and Bau *et al.* (2017), present a promising alternative approach. These methods aim to explain an agent’s decision-making process in terms of user-defined concepts, departing from the pixel-level focus of visual explanations.

While symbolic explanations are gaining traction, they’ve primarily been applied to one-shot classification problems. However, in this thesis, we address the challenge of generating explanations for deep reinforcement learning-based agents that tackle sequential decision-making problems. There have been only a few efforts in this direction thus far. One noteworthy example is the work of Hayes and Shah (2017), which utilizes high-level concepts to create policy summaries. These summaries consist of logical formulas specifying the conditions under which the agent takes particular actions. While valuable, these summaries may not fully address contrastive queries like ”Why did the agent take action A instead of B”, which are common in human-AI interactions (Miller (2019)). Our developed techniques can answer such queries posed by the users.

Other noteworthy efforts in explaining deep reinforcement learning-based systems also include van der Waa *et al.* (2018) and Madumal *et al.* (2020). Both techniques aim to answer users’ contrastive queries. The explanations to these queries essentially emphasize the difference in outcomes between the original plan and the user-suggested alternative, expressed in terms of user-defined concepts. However, it’s essential to recognize that both approaches introduce additional requirements to the domain specifications. Specifically, van der Waa *et al.* (2018) necessitates that the domain designer assigns positive and negative outcomes to each action, while Madumal *et al.* (2020) requires specifying the domain dynamics as a structural causal model using user-defined concepts. In this thesis, we tackle the challenge of generating explanations for deep RL based agents while not imposing any additional constraints on the how

the task is specified.

While letting the deep RL system explain their behavior is challenging, it is equally challenging for the user to communicate with such agents. For an effective human-robot interaction, the human must be able to instruct the agent on how it must behave so that the agent’s behavior is aligned with the user’s preference. Users are more likely to rely on and have confidence in a robot that consistently respect their choices and values. In the next section, we will illustrate current approaches that allow users to express their preferences to the agent, noting that most of these techniques rely on communication directly through videos of the agent’s behavior rather than symbolic representations.

2.3 Expressing User’s Preferences to Agents with Inscrutable Task Representation

When it comes to instructing agents with non-interpretable task representations, there are primarily two main approaches: learning from evaluative feedback and preference-based reinforcement learning (PbRL).

In the category of learning from evaluative feedback, representative works include Knox and Stone (2009) and MacGlashan *et al.* (2017) that were initially designed for low-dimensional state spaces. However, extensions of these techniques, such as Warnell *et al.* (2018), Cui and Niekum (2018), and Arumugam *et al.* (2019), have adapted them for pixel-level or high-dimensional state spaces. The core idea behind these approaches involves a human observer in the loop who observes the agent’s interactions in the environment. The observer provides positive feedback for desirable actions and negative feedback for undesirable ones. This feedback is then used to approximate the value of the state (Knox and Stone (2009)), an advantage estimate (MacGlashan *et al.* (2017)), or directly determine the user’s latent reward model (Cui and Niekum (2018)). Standard reinforcement learning techniques, including policy

gradient methods, are subsequently applied to train a policy that aligns with the human’s preferences. It’s important to note that this feedback can be demanding for users, as they need to assign values to behaviors based on their expected impact on future behavior.

Another approach for learning user preferences is preference-based reinforcement learning (PbRL) (Christiano *et al.* (2017), Lee *et al.* (2021), Bukharin *et al.* (2023)). These methods present users with pairs of trajectories and ask them to rank these trajectories based on their preference. The agent then learns a reward function that can produce similar rankings to those provided by the user. This reward function approximates the user’s latent reward model for the task, and standard reinforcement learning techniques are used to learn the user’s preferred policy. PbRL approaches are particularly well-suited for tacit knowledge tasks where user preferences cannot be easily expressed in symbolic terms. However, we posit that such approaches would not scale well in complex sequential decision making problems that involve both tacit and explicit knowledge as exchanging information solely over trajectory comparison can induce high cognitive load for humans and ultimately be a very weak training signal.

In this thesis, we advocate for the utilization of symbolic specification of user preferences. In our proposed human-AI interaction system, we provide a shared symbolic vocabulary that is both interpretable to users and grounded in the agent’s task representation. Users can employ this shared vocabulary to express their preferences symbolically. Our primary focus in this thesis is on the development of feedback-efficient techniques that facilitate the real-time addition of new terms to this shared vocabulary, thereby accommodating the potentially distinct preferences of each current user.

GENERATING EXPLANATIONS FOR AGENT WITH INSCRUTABLE TASK REPRESENTATION

We start our discussion of the human-AI interaction techniques developed in this thesis for interpretable and configurable agents by addressing situations in which the agent must explain its decision to the user, even though it relies on an internal representation of the task that the user cannot interpret. Such representations are common in modern AI agents, particularly those utilizing high-dimensional task representations learned through neural networks. Prior research on making such agents more interpretable has mainly centered on one-shot classification tasks, using approaches like highlighting input regions salient to the classification result (Sundararajan *et al.* (2017), Selvaraju *et al.* (2016)).

In contrast, our primary focus is on agents dealing with sequential decision-making tasks. These tasks require reasoning about the dynamics of the environment, and the explanation techniques typically employed for one-shot classification tasks are insufficient for generating explanations suitable for such agents.

In our case of inscrutable task models, relying on the agent’s internal representation to explain its actions isn’t feasible. One solution could be altering the task representation itself to make it symbolic. However, this approach raises concerns about the overall effectiveness and versatility of such agents, as there isn’t a consensus on whether symbolic or non-symbolic techniques are more effective. Therefore, in this study, we design agents that only compute explanations in symbolic terms that are comprehensible to humans while continuing to reason using its original represen-

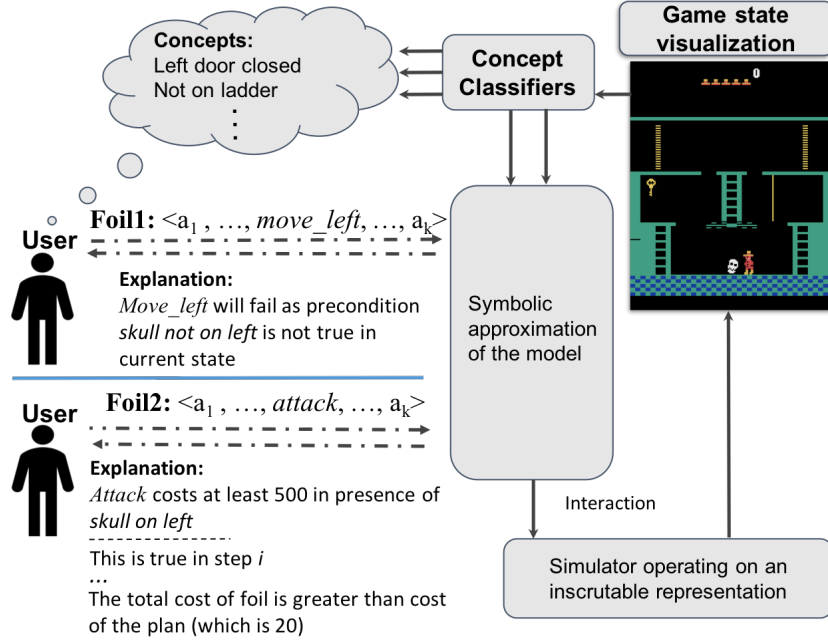


Figure 3.1: The explanatory dialogue starts when the agent’s plan incorporates jumping over the skull, prompting the user to propose specific alternative plans. The system then explains its preference for the original plan using model information in terms of propositional concepts that the user understands. These concepts themselves are grounded in the agent’s task representation as classifiers.

tation.

Accordingly, in this work, we introduce a framework that uses a predefined user vocabulary as a symbolic interface between the human and the AI agent. The agent uses terms from this vocabulary to explain its decisions to the user. We specifically address explanations required to deal with contrastive queries commonly encountered in human-AI interactions (Miller, 2019), which take the form of ”why P (the system’s proposed decision) and not Q (the user’s proposed alternative)?” To generate explanations for these queries, we create a local symbolic model approximating the task’s dynamics, similar to a PDDL specification (Aeronautiques *et al.*, 1998). The models

we learn describe actions using preconditions and action costs as functions of concepts true in the state in which the action occurs. With these model components, we can justify why the user’s alternative is either invalid or more costly than the agent’s original plan. Figure 3.1 presents the flow of the proposed explanation generation process in the context of an Atari game named Montezuma’s Revenge (Wikipedia contributors, 2019).

We will begin describing our explanatory system by first going over our specific problem setting detailing the agent’s task formulation for generating its plans and then describing the symbolic models used as approximations of the task representation. We’ll demonstrate how the components required for explanations are learned through agent sampling experiences from the agent’s model. Additionally, we’ll define the concept of explanatory confidence, which measures the quality of explanations, aiding users in calibrating their trust in the agent’s explanations. Finally, we’ll present a series of evaluations, including computational experiments and human subject studies conducted in two domains.

3.1 Problem Setting

Our setup comprises an agent, whether it is programmed or based on reinforcement learning (RL), which possesses a model of the task’s dynamics that is inscrutable to the user. In other words, the user cannot directly utilize the model representation used by the agent. It’s important to note that the term “model” is used broadly here, encompassing tabular models defined over extensive atomic state spaces, neural network-based dynamics models potentially learned over latent state representations, and even simulators.

The sole constraint imposed on the model is that we can generate potential experiences from it. Irrespective of the underlying representation, we denote the model as

the tuple $\mathcal{M} = \langle S, A, T, \mathcal{C}, \mathbb{G}, I \rangle$. In this tuple, S and A denote the state and action sets, respectively. The transition function $T : S \times A \rightarrow S \cup \perp$ denotes the transition model, where $\perp$ signifies an absorbing state reached when the agent takes an invalid action. The function $\mathcal{C} : S \times A \rightarrow \mathbb{R}$ captures the cost associated with actions. We are concerned with goal-directed agents whose objective is to steer the world’s state towards one of the goal states in the set $\mathbb{G}$ from an initial state I . The solution takes the form of a sequence of actions or a plan denoted as π .

To facilitate explanations, we employ symbolic action models with preconditions and cost functions, akin to PDDL models (Aeronautiques *et al.*, 1998). Such a model can be represented by the tuple $\mathcal{M}_{\mathcal{S}} = \langle F_{\mathcal{S}}, A_{\mathcal{S}}, I_{\mathcal{S}}, G_{\mathcal{S}}, \mathcal{C}_{\mathcal{S}} \rangle$. In this representation, $F_{\mathcal{S}}$ comprises of propositional state variables or fluents defining the state space, $A_{\mathcal{S}}$ is the set of actions, $I_{\mathcal{S}}$ represents the initial state, and $G_{\mathcal{S}}$ defines the goal specification. Each valid state in the problem is uniquely identified by the subset of state variables (or fluents) that are true in that state i.e. for any state $s \in S_{\mathcal{M}_{\mathcal{S}}}$, where $S_{\mathcal{M}_{\mathcal{S}}}$ is the set of states for $\mathcal{M}_{\mathcal{S}}$, it holds that $s \subseteq F_{\mathcal{S}}$. Importantly, each fluent in $F_{\mathcal{S}}$ corresponds to some term in the symbolic vocabulary shared with the user. Furthermore, each action $a \in A_{\mathcal{S}}$ is characterized by its preconditions $prec_a$, specifying the set of fluents that must be true in a state for action a to be executable, and the effects of executing the action as fluents that would become true once the action is executed. We denote the state resulting from executing action a in state s as $a(s)$.

Our focus is on models where the preconditions are represented as a conjunction of propositions. If an action is executed in a state with missing preconditions, the execution leads to the invalid state $\perp$. In contrast to standard PDDL models, where the cost of executing an action is independent of states, we use a state-dependent cost function of the form $\mathcal{C}_{\mathcal{S}} : 2^F \times A_{\mathcal{S}} \rightarrow \mathbb{R}$ to capture forms of cost functions commonly found in RL benchmarks.

3.2 Contrastive Explanations

In our specific explanatory scenario, as depicted in Figure 3.1, the agent devises a plan π to achieve one of the goals specified in $\mathbb{G}$, starting from the initial state I . The user responds by suggesting an alternative plan π_f (referred to as "the foil") that they believe should be followed instead. In this context, the system's task is to explain why π might be preferable over π_f .

The system can accomplish this by demonstrating that either π_f is invalid (i.e., it does not lead to a goal state or one of the actions in π_f results in the invalid state $\perp$), or π_f is more costly than π (i.e. $\mathcal{C}(I, \pi) < \mathcal{C}(I, \pi_f)$).

To illustrate this interaction, consider a modified version of Montezuma's Revenge (see Figure 3.1). In this scenario, the agent begins from the highest platform, and the objective is to reach the key. The suggested plan π may involve the agent descending to the lowest level, jumping over a skull, and reaching the key at a total cost of 20. The user proposes two possible foils, both resembling π but employing different strategies instead of jumping over the skull. *Foil1* suggests the agent simply moves left (i.e., attempting to move "through" the skull), while *Foil2* suggests the agent performs an **attack** action (not originally part of the game) before moving on to the key.

Utilizing its internal model, the system can recognize that in the first case, moving left would lead to an invalid state, and in the second case, the foil is costlier. Effectively conveying this information to the user poses a challenge. If a shared visual communication channel exists, the agent could attempt to demonstrate the outcomes of these alternate strategies. However, this might burden the user with additional cognitive load and potential confusion, as they may struggle to discern why, in a specific state, the "move left" action was invalid and why the "attack" action was costly.

As revealed in our user study and noted by Atrey *et al.* (2019), even highlighting visual features in saliency maps may not adequately resolve this confusion. Therefore, this situation calls for methods capable of expressing possible explanations in terms that the user can comprehend.

3.2.1 Learning Concept Maps

Our system takes as input *the set of propositional concepts the user associates with the task*. These form the set of fluents in $\mathcal{M}_S$. For Montezuma’s Revenge, these concepts could include notions such as *agent being on a ladder*, *holding onto the key*, or *being next to the skull*. Each concept corresponds to a propositional fact that the user links to the task’s states, believing that the presence or absence of these facts in a state can influence the dynamics and the cost function. These concepts can be gathered from subject matter experts, as demonstrated by Cai *et al.* (2019). Alternatively, the user can interact with or observe the agent and then provide a set of concepts. In our evaluation of the Sokoban domains (section 3.5) we used the latter approach to collect the propositions.

Each concept is grounded as a binary classifier, determining whether the proposition is present or absent in a given internal state. This approach allows us to transform atomic states into a factored representation. Let $\mathbb{C}$ represent the set of classifiers corresponding to these high-level concepts. For a state $s \in S$, we use the notation $\mathbb{C}(s)$ to specify the concepts that are true in s , denoted as $\mathbb{C}(s) = \{c_i | c_i \in \mathbb{C} \wedge c_i(s) = 1\}$ (where c_i refers to the classifier corresponding to the i^{th} concept, and we also use this notation for the name of the i^{th} concept). The training data for such concept classifiers can be sourced from the user, who provides sets of positive and negative examples for each concept. Classifiers can then be trained using either the model states or the internal representations employed by the agent’s decision-making pro-

cess, such as activations of intermediate neural network layers. In chapter 5, we see how we can efficiently collect the positive and negative examples of these concepts from the user to make the concept grounding process more feedback efficient.

3.2.2 Symbolic explanation using concepts

To explain the preference of plan π over foil π_f , we will provide the user with model information derived from a symbolic representation of the agent’s model. Instead of requiring this model to be an exact representation of the complete agent model, we will focus on accurately capturing a subset of the model by aiming to learn a local approximation. Here’s a formal definition of the approximate symbolic model:

Definition 1. A symbolic model $\mathcal{M}_S^{\mathbb{C}} = \langle \mathbb{C}, A_S^{\mathbb{C}}, \mathbb{C}(I), \mathbb{C}(\mathbb{G}), \mathcal{C}_S^{\mathbb{C}} \rangle$ is considered a local symbolic approximation of the model $\mathcal{M}^R = \langle S, A, T, \mathcal{C} \rangle$ for regions of interest $\hat{S} \subseteq S$ if, for every state $s \in \hat{S}$ and every action $a \in A$, there exists an equivalent action $a^{\mathbb{C}} \in A_S^{\mathbb{C}}$ such that:

$$(a) \ a^{\mathbb{C}}(\mathbb{C}(s)) = \mathbb{C}(T(s, a)) \text{ (assuming } \mathbb{C}(\perp) = \perp),$$

$$(b) \ \mathcal{C}_S^{\mathbb{C}}(\mathbb{C}(s), a) = \mathcal{C}(s, a), \text{ and}$$

$$(c) \ \mathbb{C}(\mathbb{G}) = \bigcap_{s_g \in \mathbb{G} \cap \hat{S}} \mathbb{C}(s_g).$$

This symbolic model resembles a PDDL-style model with preconditions and conditional costs defined over a conjunction of positive propositional literals. Importantly, it’s worth noting that establishing the preference of the plan does not necessitate informing the users about the entire model $\mathcal{M}_S^{\mathbb{C}}$, but only the relevant parts. We will now see which model components will suffice for an explanation for preference of π over foil π_f .

To demonstrate the invalidity of π_f , we need to simply explain the failure of the first action, denoted as a_i , that results in an invalid state (for example, for *Foil1*,

this corresponds to the *move-left* action at the state visualized in Figure 3.1). We explain the failure of this action in a state by identifying a proposition that serves as an action precondition and is absent in that state. Therefore, a concept $c_i \in \mathbb{C}$ is considered an explanation for the failure of action a_i at state s_i if $c_i \in \text{prec}_{a_i} \setminus \mathbb{C}(s_i)$. For instance, for *Foil1*, the explanation would be: ***the action move-left failed in the state as the precondition skull-not-on-left was false in the state.***

This formulation also allows us to capture the failure to achieve a goal by adding an additional goal action at the end of the plan, causing the state to transition to an absorber end state. This action fails for all states except those in $\mathbb{G}$. Notably, we focus on identifying a single precondition rather than identifying all missing preconditions, as research in social sciences has shown that selectivity or minimality is an essential property of effective explanations (Miller, 2019).

For explaining suboptimality of π_f , we provide the user with information about the symbolic cost function $\mathcal{C}_S^{\mathbb{C}}$. To ensure brevity, instead of presenting the entire cost function, we aim to learn and provide an abstraction of the cost function, denoted as $\mathcal{C}_s^{abs}$ defined as follows:

Definition 2. For the symbolic model $\mathcal{M}_S^{\mathbb{C}} = \langle \mathbb{C}, A_S^{\mathbb{C}}, \mathbb{C}(I), \mathbb{C}(\mathbb{G}), \mathcal{C}_S^{\mathbb{C}} \rangle$, an abstract cost function $\mathcal{C}_S^{abs} : 2^{\mathbb{C}} \times A_S^{\mathbb{C}} \rightarrow \mathbb{R}$ is specified as: $\mathcal{C}_S^{abs}(\{c_1, \dots, c_k\}, a) = \min\{\mathcal{C}_S^{\mathbb{C}}(s, a) \mid s \in S_{\mathcal{M}_S^{\mathbb{C}}} \wedge \{c_1, \dots, c_k\} \subseteq s\}$.

Intuitively, $\mathcal{C}_S^{abs}(\{c_1, \dots, c_k\}, a) = k$ can be understood as stating that executing the action a , in the presence of concepts $\{c_1, \dots, c_k\}$ costs at least k . To explain suboptimality, we identify a sequence of concept sets $\mathbb{C}_{\pi_f} = \langle \hat{\mathbb{C}}_1, \dots, \hat{\mathbb{C}}_k \rangle$, corresponding to each step of the foil $\pi_f = \langle a_1, \dots, a_k \rangle$. These sets must satisfy two conditions:

- (a) $\hat{\mathbb{C}}_i$ is a subset of concepts in the corresponding state reached by the foil.

- (b) The total cost according to the abstract cost function, defined over the concept subsets, exceeds the cost of the original plan. In other words,

$$\sum_{i=\{1..k\}} \mathcal{C}_S^{abs}(\hat{\mathbb{C}}_i, a_i) > \mathcal{C}(I, \pi)$$

For *Foil2*, the explanation would include the information – ***executing the action attack in the presence of the concept skull-on-left, will cost at least 500.***

3.3 Identifying Explanations Through Sample-Based Trials

For identifying the model parts, we will rely on collecting experience from agent’s internal model ($\mathcal{M}_S$) to build symbolic estimates. Since we can separate the two explanation cases of invalidating the foil or proving the foil’s suboptimality, using the agent’s internal model, we will only focus on the problem of identifying the model parts given the required explanation type.

3.3.1 Identifying failing precondition

In the context of explaining invalid foils, our goal is to identify a concept that is absent in the failing state s_{fail} but is a precondition for the action a_{fail} in the foil. We approximate whether a concept is a precondition by checking whether they appear in all the states sampled from the model where a_{fail} is executable. Let $\mathbb{S}$ denote the set of all sampled states. We rely on a sampler (referred to as **Sampler**) to create the set $\mathbb{S}$, and the number of samples used is limited by a sampling budget ℓ . The set $\mathcal{H}_{\mathbb{P}}$ captures the set of hypotheses regarding the missing precondition maintained during the search. We present the pseudocode for finding such preconditions in Algorithm 1. The search starts with a set of hypotheses for preconditions $\mathcal{H}_{\mathbb{P}}$ and rejects individual hypotheses whenever the search encounters a state where the action a_{fail} is executable, but the concept is absent. The worst-case time complexity of this search is linear with

Algorithm 1: Algorithm for finding missing preconditions

Input : $s_{fail}, a_{fail}, \text{Sampler}, \mathcal{M}^R, \mathbb{C}, \ell$

Output: Missing precondition C_{prec}

$\mathcal{H}_{\mathbb{P}} \leftarrow$ Missing concepts in s_{fail} ;

sample_count = 0;

while sample_count < ℓ **do**

$s \sim \text{Sampler}$;

if $T(s, a_{fail}) \neq \perp$ **then** $\mathcal{H}_{\mathbb{P}} \leftarrow \mathbb{C}(s) \cap \mathcal{H}_{\mathbb{P}}$ /* Remove hypothesis if the
 concept is absent in the state where a_{fail} is executable */

if $|\mathcal{H}_{\mathbb{P}}| = 0$ **then return** signal that concept list is insufficient

 sample_count += 1

end

return any $C_i \in \text{poss_prec_set}$

respect to the sampling budget.

To ensure that the learned symbolic model components are local approximations of the agent’s internal model, we consider only samples within some distance from the states in the plan and the foil. For reachability-based distances, random walks can be employed to generate the samples. Specifically, we can initiate a random walk from one of the states observed in the plan/foil and conclude the sampling episode when the number of steps taken exceeds a predefined threshold.

Note that if the hypotheses set becomes empty during the search, it indicates that the current set of concepts is insufficient to explain the given plan failure. Our focus on learning a single model component simplifies the learning problem and, as discussed in Section 3.4, allows for quantifying uncertainty related to the learned model components and accommodating for noisy concept classifiers.

3.3.2 Identifying cost function

Algorithm 2: Algorithm for finding cost function

Input : $\pi_f, \mathcal{C}_\pi, \text{Sampler}, \mathcal{M}^R, \mathbb{C}, \ell$

Output: Abstract cost function $\mathbb{C}_{\pi_f}$

```

for conc_limit in 1 to  $|\mathbb{C}|$  do
    current_foil_cost = 0, conc_list = [ ],  $s \leftarrow I$  ;
    for  $i$  in 1 to  $k$  (length of foil) do
         $\hat{\mathbb{C}}_i, \text{min\_cost} = \text{find\_min\_conc\_set}(\mathbb{C}(s), a_i, \text{conc\_limit}, \ell)$  ;
        current_foil_cost += min_cost ;
        conc_list.push( $\hat{\mathbb{C}}_i, \text{min\_cost}$ ) ;
         $s \leftarrow T(s, a_i)$  ;
    end
    if current_foil_cost >  $\mathcal{C}_\pi$  then return conc_list;
end
return signal that the concept list is incomplete

```

We will employ a similar sampling-based method to identify the cost function abstraction. Unlike the precondition failure case, there is no single action we can choose, but rather we need to choose a level of abstraction for each action in the foil (as explained in Section 3.2.2). However, it may be possible in many cases to explain the suboptimality of the foil by only referring to a subset of actions in the foil.

Our aim is to find cost abstractions for each action in the foil to explain its suboptimality. The greedy algorithm, depicted in Algorithm 2, helps us achieve this. The procedure, labeled as `find_min_conc_set`, takes the current concept representation of each state s_i in the foil and searches for the subset $\hat{\mathbb{C}}_i$ of $\mathbb{C}(s_i)$ with the maximum value for $\mathcal{C}_S^{abs}(\hat{\mathbb{C}}_i, a_i)$. As mentioned in Definition 2, the value of $\mathcal{C}_S^{abs}(\hat{\mathbb{C}}_i, a_i)$ is the

minimal cost observed when executing the action a_i in a state where $\hat{\mathbb{C}}_i$ is true. This value is approximated through sampling (with budget ℓ), and the subset size is upper bounded by ‘conc_limit’. The algorithm incrementally increases the ‘conc_limit’ value until a solution is found or if it exceeds the vocabulary set size. The worst-case time complexity of this search is $\mathcal{O}(|\mathbb{C}| \times |\pi_f| \times |\ell|)$.

Note that while this algorithm is not necessarily optimal, it has proven to be effective for the scenarios we tested. Similar to the previous case, we can enforce the required locality within the sampler. If we are unable to identify a valid explanation when ‘conc_limit’ is set to the vocabulary size, it indicates the insufficiency of current vocabulary to explain the suboptimality.

3.4 Quantifying Explanatory Confidence

When dealing with post hoc explanations, a critical challenge is ensuring the fidelity of the generated explanations and quantifying how accurately they reflect the agent’s actual decision-making process. In our case, we are particularly interested in judging whether the model components learned using the algorithms described in Section 3.3 are part of the exact symbolic representation $\mathcal{M}_S^{\mathbb{C}}$ for a given set of states $\hat{S}$. Since we are identifying model components based on concepts provided by the user, there’s a risk of generating explanations that align with the user’s existing biases about the task.

To address this challenge, we introduce a confidence value that associates a probability of correctness with each learned model component. This confidence value is derived by leveraging learned relationships between the concepts. Additionally, our confidence measure takes into account the fact that the learned concept classifiers may be noisy. We treat the outputs of the classifiers as noisy observations of the underlying concepts. Specifically, we define a probabilistic model $P(O_{c_i}^s | c_i \in S)$, where $O_{c_i}^s$

indicates whether the concept was observed in state s and $c_i \in S$ represents whether the concept was actually present in the state.

3.4.1 Graphical model for computing confidence in explanations

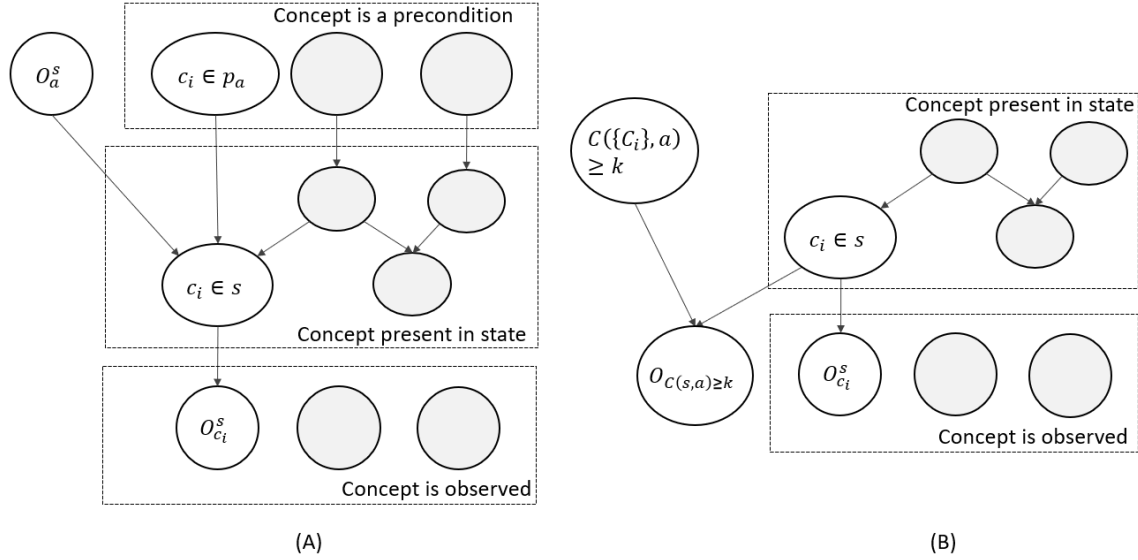


Figure 3.2: Simplified probabilistic graphical models for confidence of learned (A) Preconditions and (B) Cost.

Given the following notation & their associated events:

- O_a^s : action a is executable in state s
- $O_{c_i}^s$: concept c_i is observed in state s
- $O_{C(s,a) \geq k}$: action a in state s costs greater than or equal to k
- $O_{C(s) \setminus c_i}$: concepts other than c_i observed in state s
- $C_s^{abs}(\{c_i\}, a) \geq k$: if c_i is true, then performing action a costs at least k
- $c_i \in p_a$: concept c_i is a precondition of a

We define the graphical model presented in Figure 3.2 (annotated with notations inline) for measuring the posterior of the predicted model component being true. Specifically, for each explanation type, we compute the following posterior probabilities:

1. $P(c_i \in p_a | O_{c_i}^s = x \wedge O_a^s \wedge O_{\mathbb{C}(s) \setminus c_i})$
2. $P(\mathcal{C}_s^{abs}(\{c_i\}, a) \geq k | O_{c_i}^s = x \wedge O_{\mathcal{C}(s,a) \geq k} \wedge O_{\mathbb{C}(s) \setminus c_i})$

3.4.2 Posterior probability of a concept being a precondition

To simplify our probability calculation, we assume that the distribution of all non-precondition concepts in states where the action is executable is the same as their overall distribution (which can be empirically estimated). This implies that the likelihood of seeing a non-precondition concept in a sampled state where the action is executable is equal to the likelihood of it appearing in any sampled state. In the most general case, this equals $P(c_i \in s | O_{\mathbb{C}(s) \setminus c_i})$, i.e., the likelihood of seeing this concept given the other concepts in the state.

Given this assumption, we can derive the posterior for concept being a precondition given that we observed c_i in state s and we found that a is executable in s i.e. $P(c_i \in p_a | O_{c_i}^s \wedge O_a^s \wedge O_{\mathbb{C}(s) \setminus c_i})$.

We know that, $P(c_i \in p_a | O_{c_i}^s \wedge O_a^s \wedge O_{\mathbb{C}(s) \setminus c_i}) = (1 - P(c_i \notin p_a | O_{c_i}^s \wedge O_a^s \wedge O_{\mathbb{C}(s) \setminus c_i}))$. Using Bayes rule, we have:

$$\begin{aligned} & P(c_i \notin p_a | O_{c_i}^s \wedge O_a^s \wedge O_{\mathbb{C}(s) \setminus c_i}) \\ &= \frac{P(O_{c_i}^s | c_i \notin p_a \wedge O_a^s \wedge O_{\mathbb{C}(s) \setminus c_i}) * P(c_i \notin p_a | O_a^s \wedge O_{\mathbb{C}(s) \setminus c_i})}{P(O_{c_i}^s | O_a^s \wedge O_{\mathbb{C}(s) \setminus c_i})} \end{aligned}$$

As per the graphical model in Figure 3.2 we know that $c_i \notin p_a$ is independent of O_a^s and $O_{\mathbb{C}(s) \setminus c_i}$. Therefore, we have $P(c_i \notin p_a | O_a^s \wedge O_{\mathbb{C}(s) \setminus c_i}) = P(c_i \notin p_a)$ where $P(c_i \notin p_a)$ is the prior probability of c_i not being precondition of action a .

We now expand $P(O_{c_i}^s | c_i \notin p_a \wedge O_a^s \wedge O_{\mathbb{C}(s) \setminus c_i})$ by marginalizing over $P(c_i \in \mathbb{C}(s))$ & using graphical model (Figure 3.2) as follows:

$$\begin{aligned} P(O_{c_i}^s | c_i \notin p_a \wedge O_a^s \wedge O_{\mathbb{C}(s) \setminus c_i}) = \\ P(O_{c_i}^s | c_i \in \mathbb{C}(s)) * P(c_i \in \mathbb{C}(s) | c_i \notin p_a \wedge O_a^s \wedge O_{\mathbb{C}(s) \setminus c_i}) + \\ P(O_{c_i}^s | c_i \notin \mathbb{C}(s)) * P(c_i \notin \mathbb{C}(s) | c_i \notin p_a \wedge O_a^s \wedge O_{\mathbb{C}(s) \setminus c_i}) \end{aligned}$$

Where as per assumption 1, we know that $P(c_i \in \mathbb{C}(s) | c_i \notin p_a \wedge O_a^s \wedge O_{\mathbb{C}(s) \setminus c_i}) = P(c_i \in \mathbb{C}(s) | O_{\mathbb{C}(s) \setminus c_i})$ & $P(c_i \notin \mathbb{C}(s) | c_i \notin p_a \wedge O_a^s \wedge O_{\mathbb{C}(s) \setminus c_i}) = P(c_i \notin \mathbb{C}(s) | O_{\mathbb{C}(s) \setminus c_i})$ which can be both determined empirically.

Finally, we can expand the denominator, by marginalizing over $c_i \notin \mathbb{C}(s)$ which gives us

$$\begin{aligned} P(O_{c_i}^s | O_a^s \wedge O_{\mathbb{C}(s) \setminus c_i}) = \\ P(O_{c_i}^s | c_i \notin p_a \wedge O_a^s \wedge O_{\mathbb{C}(s) \setminus c_i}) * P(c_i \notin p_a) \\ + P(O_{c_i}^s | c_i \in p_a \wedge O_a^s \wedge O_{\mathbb{C}(s) \setminus c_i}) * P(c_i \in p_a) \end{aligned}$$

Where $P(O_{c_i}^s | c_i \notin p_a \wedge O_a^s \wedge O_{\mathbb{C}(s) \setminus c_i}) * P(c_i \notin p_a)$ was expanded earlier and $P(O_{c_i}^s | c_i \in p_a \wedge O_a^s \wedge O_{\mathbb{C}(s) \setminus c_i}) * P(c_i \in p_a)$ can be similarly expanded:

$$\begin{aligned} P(O_{c_i}^s | c_i \in p_a \wedge O_a^s \wedge O_{\mathbb{C}(s) \setminus c_i}) = \\ P(O_{c_i}^s | c_i \in \mathbb{C}(s)) * P(c_i \in \mathbb{C}(s) | c_i \in p_a \wedge O_a^s \wedge O_{\mathbb{C}(s) \setminus c_i}) + \\ P(O_{c_i}^s | c_i \notin \mathbb{C}(s)) * P(c_i \notin \mathbb{C}(s) | c_i \in p_a \wedge O_a^s \wedge O_{\mathbb{C}(s) \setminus c_i}) \end{aligned}$$

Where $P(c_i \in \mathbb{C}(s) | c_i \in p_a \wedge O_a^s \wedge O_{\mathbb{C}(s) \setminus c_i}) = 1$ & $P(c_i \notin \mathbb{C}(s) | c_i \in p_a \wedge O_a^s \wedge O_{\mathbb{C}(s) \setminus c_i}) = 0$.

Given these simplifications, we can now compute the required probability, $P(c_i \in p_a | O_{c_i}^s \wedge O_a^s \wedge O_{\mathbb{C}(s) \setminus c_i})$.

3.4.3 Posterior probability for cost function abstraction

We compute the posterior for the cost function abstraction as follows when we observe a concept c_i in a state s where action a is executable and the cost of the action is greater than k .

$$P(\mathcal{C}_s^{abs}(\{c_i\}, a) \geq k | O_{c_i}^s \wedge O_{\mathcal{C}(s,a) \geq k}) = \frac{P(O_{\mathcal{C}(s,a) \geq k}, O_{c_i}^s | \mathcal{C}_s^{abs}(\{c_i\}, a) \geq k) * P(\mathcal{C}_s^{abs}(\{c_i\}, a) \geq k)}{P(O_{\mathcal{C}(s,a) \geq k}, O_{c_i}^s)}$$

Upon marginalizing the numerator over $P(c_i \in \mathbb{C}(s))$, we get

$$\begin{aligned} & (P(O_{\mathcal{C}(s,a) \geq k}, O_{c_i}^s | c_i \in \mathbb{C}(s), \mathcal{C}_s^{abs}(\{c_i\}, a) \geq k) * P(c_i \in \mathbb{C}(s)) + \\ & \quad P(O_{\mathcal{C}(s,a) \geq k}, O_{c_i}^s | c_i \notin \mathbb{C}(s), \mathcal{C}_s^{abs}(\{c_i\}, a) \geq k) * P(c_i \notin \mathbb{C}(s))) * P(\mathcal{C}_s^{abs}(\{c_i\}, a) \geq k) \\ = & \frac{P(O_{\mathcal{C}(s,a) \geq k}, O_{c_i}^s)}{P(O_{\mathcal{C}(s,a) \geq k}, O_{c_i}^s)} \end{aligned}$$

Given their parents, $O_{\mathcal{C}(s,a) \geq k}$ and $O_{c_i}^s$ are conditionally independent, and given its parent $O_{c_i}^s$ is independent of $\mathcal{C}_s^{abs}(\{c_i\}, a) \geq k$ (see graphical model in Figure 3.2), there by giving

$$\begin{aligned} & (P(O_{\mathcal{C}(s,a) \geq k} | c_i \in \mathbb{C}(s), \mathcal{C}_s^{abs}(\{c_i\}, a) \geq k) * P(O_{c_i}^s | c_i \in \mathbb{C}(s)) * P(c_i \in \mathbb{C}(s)) + \\ & \quad P(O_{\mathcal{C}(s,a) \geq k} | c_i \notin \mathbb{C}(s), \mathcal{C}_s^{abs}(\{c_i\}, a) \geq k) * P(O_{c_i}^s | c_i \notin \mathbb{C}(s)) * P(c_i \notin \mathbb{C}(s))) * \\ & \quad P(\mathcal{C}_s^{abs}(\{c_i\}, a) \geq k) \\ = & \frac{P(O_{\mathcal{C}(s,a) \geq k}, O_{c_i}^s)}{P(O_{\mathcal{C}(s,a) \geq k}, O_{c_i}^s)} \end{aligned}$$

Now $P(O_{\mathcal{C}(s,a) \geq k} | c_i \in \mathbb{C}(s), \mathcal{C}_s^{abs}(\{c_i\}, a) \geq k) = 1$ and $P(O_{\mathcal{C}(s,a) \geq k} | c_i \notin \mathbb{C}(s), \mathcal{C}_s^{abs}(\{c_i\}, a) \geq k)$ can either be empirically estimated from true labels or we can make the assumption that is equal to $p_{\mathcal{C}(.,a) \geq k}$, the probability that executing action a in any state costs greater than k (which we made use of in our experiments). Given this, we get $P(\mathcal{C}_s^{abs}(\{c_i\}, a) \geq k | O_{c_i}^s \wedge O_{\mathcal{C}(s,a) \geq k})$ to be:

$$\begin{aligned} & (P(O_{c_i}^s | c_i \in \mathbb{C}(s))P(c_i \in \mathbb{C}(s)) + \\ & \quad p_{\mathcal{C}(.,a) \geq k} * P(O_{c_i}^s | c_i \notin \mathbb{C}(s)) * P(c_i \notin \mathbb{C}(s))) * P(\mathcal{C}_s^{abs}(\{c_i\}, a) \geq k) \\ = & \frac{P(O_{\mathcal{C}(s,a) \geq k}, O_{c_i}^s)}{P(O_{\mathcal{C}(s,a) \geq k}, O_{c_i}^s)} \end{aligned}$$

3.4.4 Modified algorithm accounting for confidence in explanations

We can now incorporate these confidence measures directly into the search algorithms described in Section 3.3 to find the explanation that maximizes these probabilities. In the case of precondition identification (algorithm 1), given a noisy classifier, we can no longer use the classifier output to directly remove a concept from the hypothesis set $\mathcal{H}_{\mathbb{P}}$. At the start of the search, we associate a prior to each hypothesis and use the observed concepts at each executable state to update the posterior of that concept being a precondition. We only remove a precondition from consideration if the probability associated with it dips below a threshold.

To obtain the cost function abstraction with the maximum probability, when invalidating a suboptimal foil, the algorithm employed is mostly the same as Algorithm 2. However, there is a difference in the case of multiple cost functions having the same upper bound in `find_min_conc_set`; in such situations, we select the one with the highest probability. All the results presented in the next section were computed using the probabilistic version of the search, which allows for the presence of noisy concept maps.

3.5 Evaluation

We evaluated our search techniques using three distinct environments: the deterministic version of Montezuma’s Revenge from the OpenAI Gym (Brockman *et al.*, 2016), and two modified versions of the Sokoban game (Schrader, 2018). To simplify our experiments, we made the assumption that the concepts are independent in the domains.

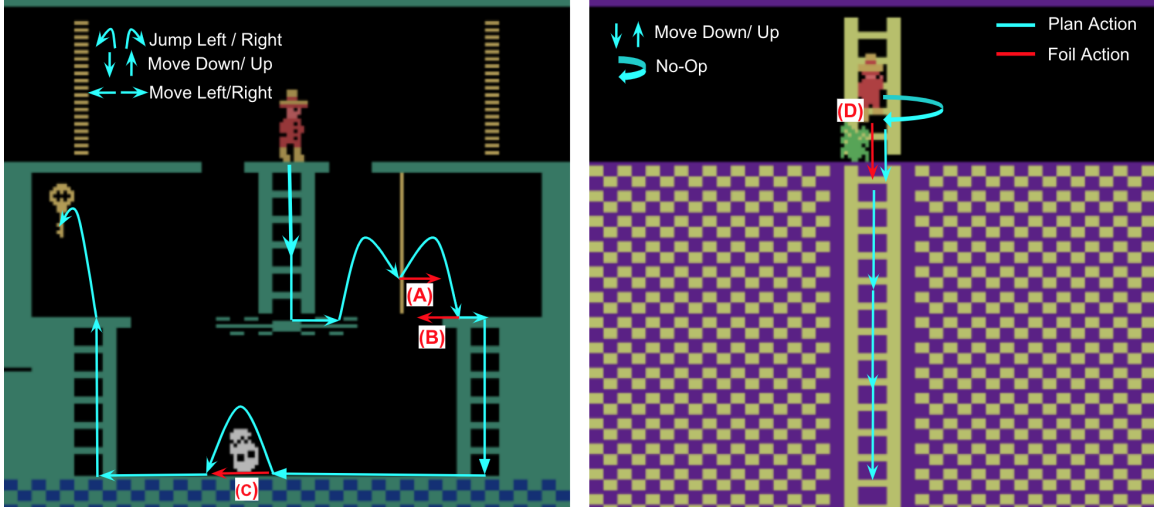


Figure 3.3: The plan and foils used in the evaluation for the game of Montezuma’s revenge in two different levels. Plans are highlighted in blue and foils in red. The concepts corresponding to missing preconditions are `not_on_rope`, `not_on_left_ledge`, `not_skull_on_left` and `is_clear_down_of_crab` for failure points A, B, C and D respectively (note all concepts were originally defined as positives and we formed the `not_` concepts by negating the output of the classifier).

Montezuma’s Revenge:

In the context of Montezuma’s Revenge, we employed a RAM-based state representation. To introduce richer preconditions , we categorized any action that didn’t alter the agent’s state or led to situations where the agent would inevitably fall to its death as failures. We handpicked four invalid foils for evaluation purposes, with three originating from screen-1 and one from screen-4 of the game (depicted in Figure 3.3). The plan for screen-1 involved the player obtaining a key, while for screen-2, the player had to reach the bottom of the screen. We defined ten concepts for each screen that we deemed relevant to the game. Subsequently, we collected positive and negative examples using automated scripts. Our concept classifiers, based on AdaBoost

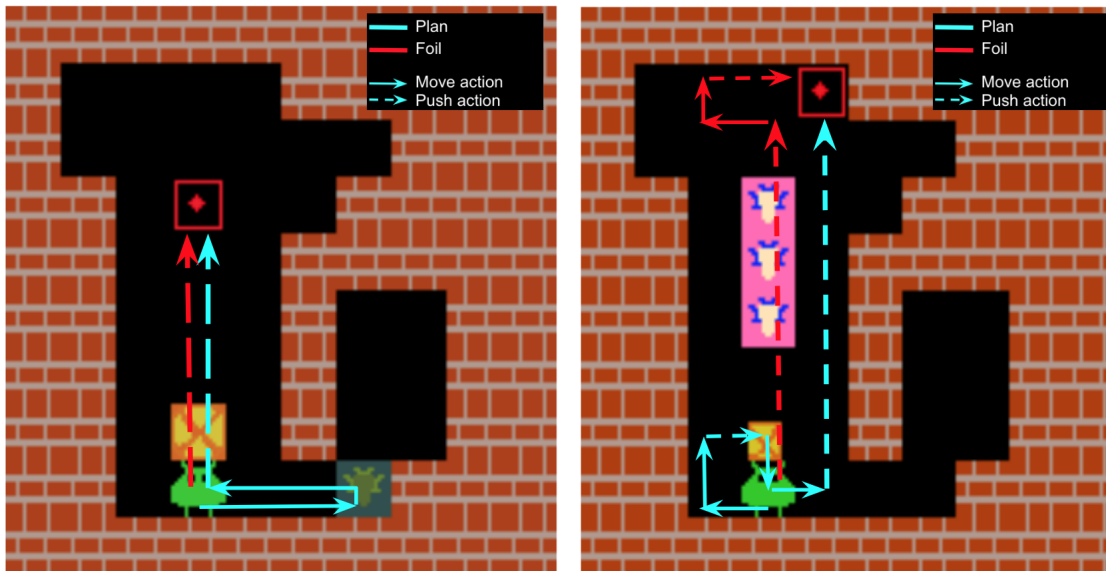


Figure 3.4: The plan and foils used in the evaluation for the two variants of the Sokoban game (sokoban-switch on the left and sokoban-cell on the right). Plans are highlighted in blue and foils in red. The concepts corresponding to the cost function, for the Sokoban cost variants are `switch_on` and `on_pink_cell` that results in push action costing 10 (instead of 1)

(Freund *et al.*, 1999), achieved an average accuracy of 99.72%.

Sokoban Variants:

We considered two variations of the traditional Sokoban game (as shown in Figure 3.4). One variant required the player to activate a switch (green cell) before pushing the box, referred to as Sokoban-switch. The other variant, Sokoban-cell, included specific cells (highlighted in pink) from which pushing the box incurred higher costs. For Sokoban-switch, we explored two subvariants: one where turning on the switch was a precondition for push actions and another where it only reduced the cost of pushing the box. The plans and foils for this domain are shown in Figure 3.4.

To identify concepts for these variants, we conducted a survey involving graduate students who were unfamiliar with our research. Participants interacted with the game via a web interface, specifically the cost-based versions for Sokoban-switch and Sokoban-cell. Following the gameplay, they were asked to describe game concepts they found relevant to particular actions. We received 25 unique concepts from six participants for Sokoban-switch and 38 unique concepts from seven participants for Sokoban-cell. User descriptions of these concepts were converted into scripts for sampling positive and negative instances. We focused on 18 concepts for Sokoban-switch and 32 concepts for Sokoban-cell, selecting them based on their frequency in game states. For concept classification, we employed Convolutional Neural Networks (CNNs) to map state images to concepts. These classifiers achieved an average accuracy of 99.46% for Sokoban-switch and 99.34% for Sokoban-cell.

3.5.1 Computational Experiments

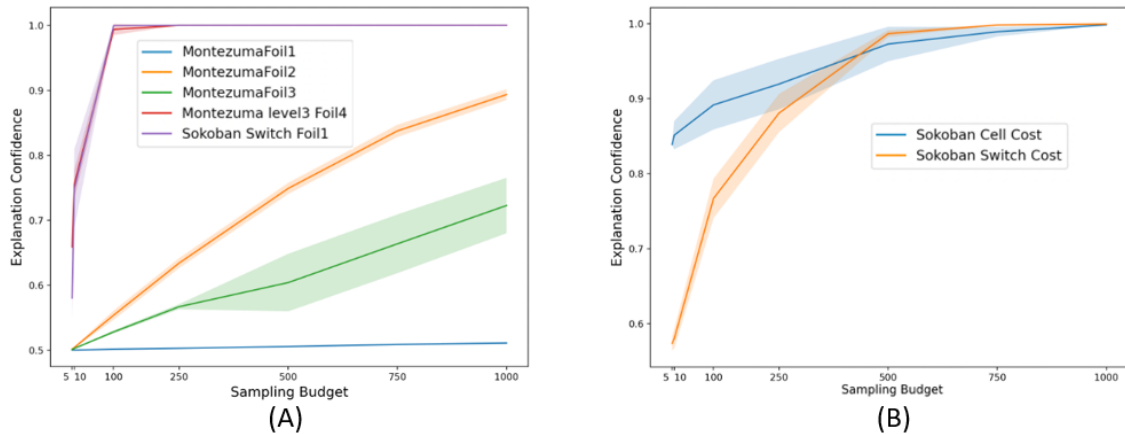


Figure 3.5: The average probability assigned to the correct model component by the search algorithms, calculated over ten random search episodes with std-deviation.

We conducted searches to identify preconditions for Montezuma’s foils and both the Sokoban-switch variants, as well as cost function based explanations for foils in

both Sokoban variants. To augment our original list of concepts, we doubled the final concept list by including negations of each concept. This resulted in 20 concepts each for Montezuma and 36 concepts for Sokoban-switch, and 64 concepts for Sokoban-cell. We calculated probabilistic models for each classifier based on their respective test sets.

For precondition identification, we executed the search with a cutoff probability of 0.01 for each concept. In each case, our search-based methods successfully identified the correct model component for explanations.

Figure 3.5(A) depicts the probability assigned by our system to the correct missing precondition as a function of the sampling budget. This data represents the average calculated across ten random episodes. Generally, the probability increases with the sampling budget, occasionally exhibiting small dips due to misclassifications. Notably, the smallest explanation likelihood was observed for `foil1` in screen 1 of Montezuma (where the foil was for the agent to move through the skull) with a value of 0.511 ± 0.001 . This is attributed to the true precondition concept, `skull-not-on-left`, being very commonly present in state space; which made its presence in the executable states not strongly supporting it as a precondition. Nevertheless, the system correctly identified this concept as the most plausible hypothesis for the precondition, as all other hypotheses were eliminated after approximately 100 samples. Similarly, Figure 3.5(B) displays the probability assigned to the identified abstract cost functions.

3.5.2 User study

With our basic explanation generation methods in place, we aimed to evaluate whether users would find such explanations helpful. Specifically, we wanted to measure the effectiveness of symbolic explanations along two dimensions:

- (a) **Preference for Symbolic Explanations:** We sought to determine whether

people preferred explanations that referred to the symbolic model components over potentially more concise and direct explanations.

- (b) **Improvement in Task Understanding:** We aimed to assess whether symbolic explanations aided users in gaining a better understanding of the task.

All participants in our study were graduate students, distinct from those who specified the concepts.

Testing preference for Symbolic Explanations

	Prefer symbolic explanations	Average Likert-score	p-value
Precondition	19/20	3.47	1.0×10^{-8}
Cost	16/20	3.21	0.03

Table 3.1: Evaluation results for testing whether participants preferred symbolic explanations indicating model components

For measuring user preference, we tested the hypothesis:

H1: *People would prefer explanations that establish the corresponding model component over ones that directly present the foil information (i.e., the failing action or the per-step cost).*

This hypothesis is particularly interesting because our baseline explanatory text is considerably more concise in terms of text size. It aligns with the selectivity principle (Miller, 2019), suggesting that people prefer explanations without extraneous information. Thus, we were effectively testing whether users found the symbolic model information extraneous.

We conducted a within-subject study design where participants were presented with an explanation generated by our method and a simple baseline. For the baseline, in the precondition case, we pointed out the failing action and the state in which it was executed. In the cost case, we presented the exact cost of executing each action in the foil. Participants were asked to choose the explanation they believed was more useful, with the order of choices randomized to ensure counterbalanced results. Additionally, they rated the completeness of the chosen explanation on a five-point Likert scale, with 1 indicating "not at all complete" and 5 indicating "complete."

For precondition case, we collected 5 responses for each Montezuma’s foil, and for cost based explanation, we obtained 10 responses for each Sokoban variant (40 in total). Table 3.1 summarizes the results from the user study, supporting H1. In fact, a binomial test demonstrates that the selections are statistically significant with respect to $\alpha = 0.05$.

Testing improvement in task understanding

Method	Average time taken	Average number of steps
Concept-Based	43.78 $\pm$ 12.59	35.87 $\pm$ 9.69
Saliency-Map based	134.24 $\pm$ 61.72	52.64 $\pm$ 11.11

Table 3.2: Evaluation results for testing whether participants preferred concept based explanations over saliency maps

To assess the effectiveness of explanations in aiding users’ understanding of the task, we examined the hypothesis:

H2: *Concept-based precondition explanations help users understand the task better than saliency map-based ones.*

Our focus was on saliency maps and preconditions, as saliency map-based explanations could potentially highlight areas corresponding to failed precondition concepts, especially when concepts align with specific regions within the image.

We measured users’ task understanding by evaluating their ability to independently complete the precondition variant of the Sokoban-switch game. We adopted a between-subject study design, where each participant was assigned to a single explanation type. Participants were instructed to complete the game within a 3-minute time limit, with bonuses offered to those who finished the game in the shortest time. They received no precise instructions about the game’s dynamics.

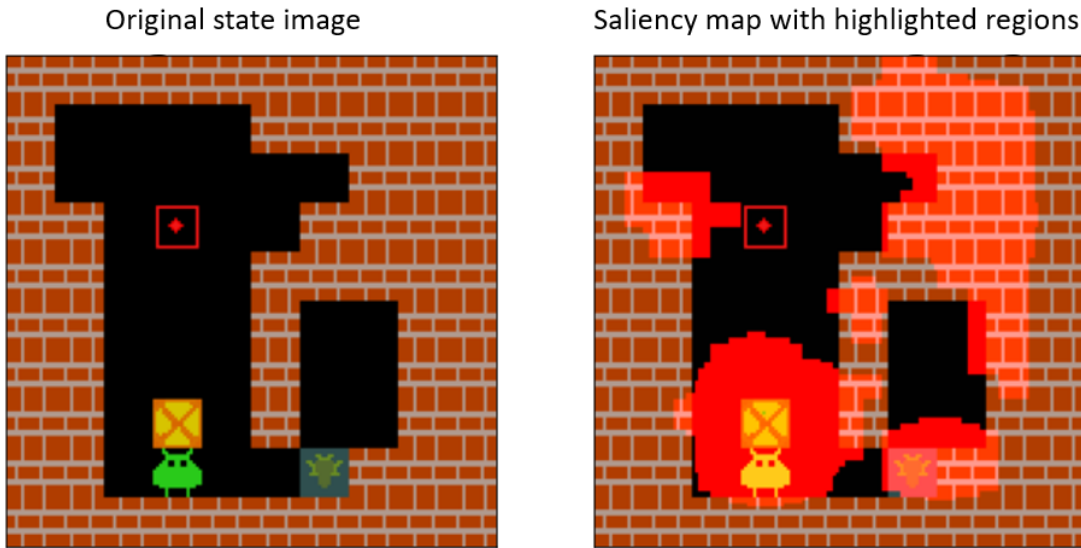


Figure 3.6: The saliency map based explanation shown to users to test hypothesis H2.

During gameplay, if a participant performed an action for which the preconditions were not met, they were shown their respective explanation. Additionally, the current episode ended, and they had to restart the game from the beginning. One group of users received precondition explanations generated through our method, while the remaining participants were presented with a saliency map. For the latter, we employed

a state-of-the-art saliency map-based explanation method Greydanus *et al.* (2018). For this technique, participants were informed that highlighted regions represented areas an AI agent would focus on if it were acting in that state. In all saliency map explanations, the highlighted region included the precondition region of the switch (as shown in Figure 3.6). We collected a total of 60 responses but had to discard 12 due to participants reporting not seeing the explanations.

Table 3.2 presents the average time taken and steps required to complete the task for the two techniques, along with 95% confidence intervals. As observed, the average values are significantly lower for concept explanations. Furthermore, a two-tailed t-test demonstrates that the results are statistically significant (p-values of 0.021 and 0.038 against a significance level $\alpha = 0.05$ for time and steps, respectively). This indicates a notable improvement in task understanding when users received concept-based precondition explanations.

3.6 Chapter Summary

In this chapter, we have developed general approaches for generating interpretable post-hoc symbolic explanations for sequential decision-making problems. Our objective is to facilitate the generation of explanations using a shared symbolic vocabulary between the user and the agent, even though the agent itself operates with non-symbolic, high-dimensional task representations. Additionally, we have introduced techniques for computing confidence in the explanations, providing users with a means to calibrate their trust in the explanation’s accuracy. These approaches have been applied across a range of domains, and we have conducted evaluations to assess their effectiveness through user studies and computational experiments. The work presented in this chapter was accepted as a paper in ICLR 2022.

GENERATING PERSONALIZED EXPLANATIONS IN SEQUENTIAL DECISION MAKING PROBLEM

We’ve discussed how agents can generate understandable explanations even when their task model is opaque to users. Clear explanations are crucial for successful human-AI interactions, but they must also align with the user’s specific background knowledge (Miller, 2019). In this chapter, we explore how our agent can craft personalized explanations tailored to the user’s mental model of the task. As illustrated in chapter 2, works done under the umbrella of *model reconciliation* (Chakraborti *et al.*, 2017) do account for user’s mental model of the task. They provide explanations that correct or update the model so that the user can correctly evaluate agent’s decisions. However, these approaches typically assume that the agent possesses the exact model of the user. Even in cases where they accommodate uncertainty about the user’s model, they require that the set of possible user models must be explicitly represented in a specific declarative form before interacting with the target user (Sreedharan *et al.*, 2018a,b).

When the agent does not have an explicitly given model of its user, the agent must learn a new model for each new user. Learning this model would require a lot of input from the user. Moreover, the agent would not be leveraging the existing knowledge it gained from learning the models of other users it has interacted with. In this work, we introduce a more practical setting in which any given task will have multiple categories of users such that users belonging to the same type would have a similar model of the task. We refer to these categories as “user types”. For example, in automated medical diagnosis, user types could be medical experts and patients.

Each type may require distinct explanations for the agent’s plans based on their differing task models. If the agent can access the models of these user types and infer the current user’s type during interaction, it can effectively explain its behavior to any user.

We explore this kind of setup in our work. However, this poses some significant challenges. For one, we may not initially know all the user types that exist. In addition, even if they are known, any target user cannot be expected to know which type they belong to. This is especially true when users are not aware of all classes of users, and how they were classified. To address these challenges, we design a system that:

1. Identifies the types of users that exist for the task and learn their model of the task
2. When interacting with the target user, infers their type and uses their model to give personalized explanations

In this chapter, we first illustrate our problem setting using a planning domain as an example. We then give background on how approximations of mental model of each type of users are learned for *model reconciliation*. We then present our computational methods for identifying the user types for a given task and learning their models. This is followed by a detailed discussion of our *partially observable Markov Decision Process* (POMDP) based formulation that we use to reason about the current user’s type and generate explanations according to its inferred type. Finally, we present a detailed evaluation of our developed techniques using both computational experiments and human subject studies.

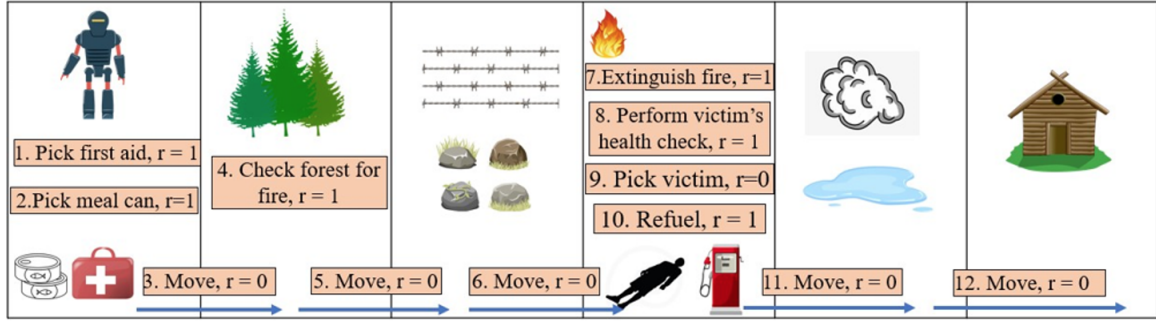


Figure 4.1: An instance of disaster-rescue domain along with a robot's execution trace with transitions labeled with their corresponding rewards

Supervisor type	Transitions with rewards, $r = 1$
Type-A	Picking meal can
	Extinguishing fire
Type-B	Picking first aid
	Passing over fence
Type-C	Checking forest for fire
	Passing over rubble
Type-D	Performing victim's health check
	Passing through smoke
Type-E	Refuel
	Passing through puddle

Table 4.1: Inaccurate model components for each possible user-type of the disaster rescue domain

4.1 Illustrative Example

We will use a disaster-rescue scenario to illustrate our techniques. This scenario also serves as one of the domains used to evaluate our solution approaches. In this domain, a rescue robot is assigned the task of evacuating a victim to a designated shelter. Additionally, the robot can perform certain actions to earn additional rewards. These actions include picking up first aid supplies, extinguishing fires, and refueling at a fuel station, all of which result in positive rewards. There are also obstacles such as rubble and puddles, but the robot is equipped to navigate through them without incurring negative rewards. In Figure 4.1, we depict an instance of this domain and present an optimal plan executed by the robot. Furthermore, the figure provides information about the reward associated with each transition.

Beyond the domain itself, there is a human supervisor responsible for evaluating the robot’s behavior. However, this supervisor may possess inaccurate knowledge about certain aspects of the domain. We distinguish five types of supervisors, denoted as *type-A* to *type-E*, each with varying levels of domain knowledge. For example, a *type-E* supervisor believes that refueling is unnecessary and associates a negative reward with it. They also assume that the robot can break when moving into a location with a puddle, leading to a negative reward in their model of the task. Table 4.1 outlines the inaccurate facts held by each supervisor type regarding the domain.

When assessing the robot’s plan, supervisors of each type may find different segments of the plan inexplicable. For instance, a *type-B* supervisor will be confused when the robot collects the first-aid supplies. Importantly, the robot is not aware of the specific supervisor types or the type of supervisor it is currently interacting with. To ensure that the entire plan is understandable to the supervisor, one straightforward approach would be to provide the entire robot’s model. However, this approach

can easily overwhelm the supervisor. Our objective is for the robot to offer only the necessary explanations to the supervisor. To achieve this, the robot must determine all the types of supervisors that exist, learn their models of the task, and, during interactions with a particular supervisor, infer their type and provide personalized explanations accordingly. In the following sections, we will delve into how our framework accomplishes this objective. To begin, we will outline the types of problems that our agent is initially designed to solve, which subsequently leads to agent’s decisions that might need to be explained to the target user.

4.2 Background

In this work, our focus is on robots that use discounted infinite horizon Markov Decision Processes (MDPs) for solving a sequential decision making problems Russell and Norvig (2005). An MDP $\mathcal{M}$ can be formally defined as tuple $\langle S, A, T, R, \gamma \rangle$, where: S is the set of states in the domain; A is the set of actions that the robot can take in any state; T is the transition function where $T(s, a, s')$ gives the probability that the robot will reach state s' after taking action a in state s ; R is the reward function such that $R(s, a, s')$ gives the reward for a transition $\langle s, a, s' \rangle$; γ is the discounting factor for the rewards obtained (where $0 \leq \gamma < 1$). A policy π for an MDP maps each state to a recommended action. The value for any state given π is the expected cumulative discounted reward obtained if the robot follows π . For a given MDP $\mathcal{M}$, the optimal policy $\pi_{\mathcal{M}}^*$ is simply the policy that would give the highest value from any state. Lastly, the Q-value function, $Q_{\pi}(s, a)$, for an MDP gives the expected value obtained if the robot takes action a in state s and then follows the policy π . We use Q^* to denote the Q-value function corresponding to the optimal policy for the MDP.

4.2.1 Learning approximate mental model of the user to generate explanations

In the context of model reconciliation as described in Chakraborti *et al.* (2017), the robot provides explanations in the form of model updates to correct the human’s understanding or model of the task. Unlike the original approach, which assumed the human’s task model was predefined, we adopt the method presented in Sreedharan *et al.* (2019) that learns this model. This method employs learned labeling models to determine the specific model information that needs to be conveyed to the user to make any transition appear explicable. This labeling model thus serves as a proxy to the user’s complete model of the task.

Following Sreedharan *et al.* (2019), we start with a set of explanatory messages denoted as $\mathbb{E} = \{e_1, e_2, \dots, e_n\}$, which contain information about the robot’s model. For instance, in the disaster-rescue domain, an example message might be “the robot receives a reward of 1.0 for picking up the first aid.”

Given a particular transition $\tau = \langle s, a, s' \rangle$, we train a labeling model, $\mathcal{L} : (T, 2^{\mathbb{E}}) \rightarrow \{0, 1\}$, where T represents the set of all possible transitions. In this model, $\mathcal{L}(\tau, \hat{\mathbb{E}}) = 1$ indicates that once the user is provided with a set of messages, $\hat{\mathbb{E}}$, they find the transition τ to be explicable. In other words, there exists an optimal policy in their updated model that can produce this transition. Conversely, $\mathcal{L}(\tau, \hat{\mathbb{E}}) = 0$ signifies that the user finds the transition inexplicable. In this case, there is no optimal policy in the user’s updated model that can lead to this transition. Thus forth, we refer to the label generated by the labeling function as the explicability label.

Importantly, unlike Sreedharan *et al.* (2019), our work acknowledges that different users of the system may possess diverse backgrounds and mental models of the task. Consequently, they may require distinct sets of explanatory messages. Our objective in this study is to explore the development of an explanatory system capable of

generating tailored explanations that cater to the specific models of individual users.

Applicability of the presented approaches for tasks with inscrutable models

Note that our work is agnostic to the task representation employed by the agent and is compatible with agents that possess opaque task representations. However, it imposes a more specific requirement than merely having a symbolic vocabulary (as in chapter 3). We assume that each message in $\mathbb{E}$ corresponds to symbolic information about the task’s dynamics, which the user can comprehend. It’s important to note that $\mathbb{E}$ does not necessarily need to encompass all possible model parameters; instead, it can be limited to a subset of parameters that a domain expert has identified as relevant to the given problem. We will now discuss the specific solution to generate personalized explanations tailored to the user’s type.

4.3 Explanation in the Presence of Multiple User Types

In this work, our main focus is on creating a system where a robot can provide explanations to users with diverse backgrounds. The explanatory system starts with the robot’s model of the task, $\mathcal{M}^r$, and a set of explanatory messages $\mathbb{E}$. To simplify the setup, we assume that each user’s understanding of a given task can be associated with one of several possible models, which in turn corresponds to one of the labeling models in a set denoted as $\mathbb{L}$. Consequently, for any user in the system of type t , we represent their labeling function as $\mathcal{L}_t \in \mathbb{L}$.

4.3.1 Learning the set of labeling models

Now the first step to setting up our explanatory system for a task will be to learn the corresponding labeling function set $\mathbb{L}$. This involves identifying all the user types and learning their respective labeling functions.

Similar to Sreedharan *et al.* (2019), we will rely on collecting data from users, which we'll refer to as *observers* to learn $\mathbb{L}$. This data includes examples that contain a transition, a set of messages, and a label indicating if the the observer will find the transition to be explicable when presented with the message set. We also assume that we know which observer generated each label. Thus, each data point takes the form $d = \langle \tau, \hat{\mathbb{E}}, l, o \rangle$, where $\tau = \langle s, a, s' \rangle$ represents a transition, $\hat{\mathbb{E}}$ is the message set, l is the corresponding explicability label, and o is a unique observer ID. Our primary objective is to group the collected data points into k clusters, such that each cluster will only contain data from a single user type. This can be followed by learning a labeling function for each type using the data from their cluster to finally get the set $\mathbb{L}$.

To begin, each observer is represented as an vector $\dot{\mathbb{E}}_o \in \{0, 1\}^{|\mathbb{E}|}$ in an embedding where each bit corresponds to a message in $\mathbb{E}$. For a given $\dot{\mathbb{E}}_o$, the set of messages whose bit is set to 1 represents the smallest set of messages which when given to the observer o will make every possible transition explicable to them. This can be thought of as the minimal set of messages that makes the human's model aligned with the robot's model of the task. To obtain $\dot{\mathbb{E}}_o$, the learner first utilizes the data points D_o collected from each observer o to train their individual labeling function $\mathcal{L}_o$. Any function approximator can be used to learn the labeling function. Then, using $\mathcal{L}_o$, we determine the smallest set of messages that would make all the transitions in D_o explicable to the observer. Finally, the computed message set serves as an approximation of $\dot{\mathbb{E}}_o$.

Once each observer has been projected in this embedding, the learner can proceed with grouping the observers using clustering techniques. In this work, we employ k-means clustering in this embedding with a specified number of clusters k . Ideally, this k should match the total number of user types N_u , which is unknown. We esti-

mate this value using the standard elbow method. To apply this method, we define a problem-specific intra-cluster variance measure which we call “disagreements”. This measure quantifies the total number of instances where two observers, clustered together and given the same set of messages, disagree on the label of a given transition. Formally, we define “disagreements” as follows:

Definition 3. *For a given clustering of observers, disagreements is equal to the total number of pairs of datapoints $d_i = \langle \tau_i, \hat{\mathbb{E}}_i, l_i, o_i \rangle$ and $d_j = \langle \tau_j, \hat{\mathbb{E}}_j, l_j, o_j \rangle$ that satisfy the following conditions: o_i and o_j belong to the same cluster, $\tau_i = \tau_j$, $\hat{\mathbb{E}}_i = \hat{\mathbb{E}}_j$, and $l_i \neq l_j$.*

We expect that the value of “disagreements” will initially decrease as we increase k from 1 up to N_u because observers belonging to different types would ideally be placed in separate clusters as the number of clusters increases. Further increasing k beyond N_u will not affect the “disagreements”. Therefore, we use the elbow point in the plot of “disagreement” vs. k as an estimate of N_u . To programmatically find this point, we use the technique given in Satopaa *et al.* (2011).

Having estimated N_u and obtained the corresponding clustering output at $k = N_u$, the learner can now learn the labeling set $\mathbb{L}$. To do so, for each cluster, we take the union of all the data points from all the observers within that cluster. Each of these sets now contains data points corresponding to an identified user type u_t . Then the labeling model $\mathcal{L}_t$ corresponding to each user type u_t is trained on the data points in their corresponding set. This gives us the desired set $\mathbb{L}$. Additionally, we learn a confidence measure that provides the confidence for the prediction made by the learned labeling model. Specifically, for each $\mathcal{L}_t$, we learn a probabilistic model, $P_{\mathcal{L}_t}(l|\tau, \hat{\mathbb{E}})$, which indicates the probability that a user of type t will assign label l to transition τ , given they have been provided with explanations $\hat{\mathbb{E}}$.

4.3.2 Explaining Behavior Traces

With the specific labeling models in place, we are now ready to generate explanations for a specific behavior trace $\Pi = \langle \tau_1, \dots, \tau_m \rangle$ where τ_i represents a transition within the trace. An explanation in this setting consists of a sequence of explanatory messages of the form $\mathcal{E} = \langle \hat{\mathbb{E}}_1, \dots, \hat{\mathbb{E}}_m \rangle$ where each $\hat{\mathbb{E}}_k$ is a set of messages that are given to the user at step k of the interaction. At each step k , the user is presented with the prefix $\langle \tau_1, \dots, \tau_k \rangle$ denoted as π_k . Whether a user of type t will find any transition τ_j to be explicable can be estimated by the function $\mathcal{L}_t(\tau_j, \bigcup_{i=1}^k (\hat{\mathbb{E}}_i))$. In other words, the user attempts to make sense of each transition using the current message as well as previous messages. The user is expected to assign labels to each transition in π_k .

Now, it's important to consider that each explanatory message comes with an associated communication cost. To facilitate analysis of the trade-offs between communication and inexplicability, we introduce the functions C_e , which quantifies the cost of presenting an explanatory message to the user, and C_{inexp} , which represents the cost of showing an inexplicable plan step to the user. Additionally, we introduce the constant λ to determine the relative weight between the total communication cost and the total inexplicability cost for the entire user interaction. Formally, the cost of an explanation sequence can be defined as follows:

Definition 4. *The cost of an explanation sequence $\mathcal{E} = \langle \hat{\mathbb{E}}_1, \dots, \hat{\mathbb{E}}_m \rangle$, for a trace Π and a user type $t \in U$ is given by*

$$C(\mathcal{E}) = \sum_{i=1}^m [\lambda \cdot C_e \cdot |\hat{\mathbb{E}}_i| + \sum_{j=1}^i (1 - \mathcal{L}_t(\tau_j, \bigcup_{k=1}^i (\hat{\mathbb{E}}_k))) \cdot C_{inexp}]$$

In our interaction scenario, if the user type were to be known, our goal would be to find an explanatory sequence that minimizes this total cost. Unfortunately, this

is not true in our case as in addition to reasoning about the explanatory sequence, we would also need to reason about the unknown user types. All we assume is that we have access to a possible prior over the user types. A useful reasoning framework that we could leverage would be that of partially observable Markov decision process (POMDP) Kaelbling *et al.* (1998). We can cast this problem of reasoning about explanation into a meta POMDP of the form $\mathcal{M}_{\mathcal{E}} = \langle S_{\mathcal{E}}, A_{\mathcal{E}}, T_{\mathcal{E}}, O_{\mathcal{E}}, \Omega_{\mathcal{E}}, \gamma_{\mathcal{E}} \rangle$. Specifically, we turn the problem of identification of explanation with unknown user type into a POMDP where the user type is the hidden part of the state. Solving a POMDP, in general, gives us a policy over the belief state. In our case, the belief state represents the likelihood that the current user belongs to one of the various learned user types. It is essentially an estimated probability distribution indicating how probable it is for the current user to be categorized into each of these learned user types.

We will now define each component of $\mathcal{M}_{\mathcal{E}}$:

- $S_{\mathcal{E}}$: In the meta-POMDP, each state is defined by several elements:
 1. A prefix, π_k , which is a segment of the trace intended to be explained.
 2. Two distinct sets of explanatory messages labeled as $\hat{\mathbb{E}}_{\text{given}}$ and $\hat{\mathbb{E}}_{\text{next}}$:
 - $\hat{\mathbb{E}}_{\text{given}}$ represents the messages already presented to the user.
 - $\hat{\mathbb{E}}_{\text{next}}$ represents the messages that will be presented to the user in the next interaction step.
 3. A sequence of binary labels denoted as L . Each binary label corresponds to a transition within the prefix and is set to 1 if the user found the transition explicable based on the messages in $\hat{\mathbb{E}}_{\text{given}}$, and 0 otherwise.
 4. The unobserved part of the state representing the user’s type, denoted as u_t .

Formally, let L_{π_k} denote the set of all possible sequence of explicability labels for the transitions within π_k and let U denote the set of all possible user types . Then the state space can be given as:

$$S_{\mathcal{E}} = \{ \langle \pi_k, L, \hat{\mathbb{E}}_{\text{given}}, \hat{\mathbb{E}}_{\text{next}}, u_t \rangle \mid k \in [1, m], L \in L_{\pi_k}, u_t \in U \\ \hat{\mathbb{E}}_{\text{given}} \in 2^{|\mathbb{E}|}, \text{ and } \hat{\mathbb{E}}_{\text{next}} \in 2^{|\mathbb{E}|} \}$$

- **$A_{\mathcal{E}}$:** Actions in the POMDP correspond to the set of explanatory messages, $\mathbb{E}$, that the robot can present to the user along with a special action called *explain*. Thus, the action space $A_{\mathcal{E}}$ can be defined as $A_{\mathcal{E}} = \mathbb{E} \cup \{\text{explain}\}$.
- **$T_{\mathcal{E}}$:** In our POMDP, there are two types of transitions depending on the action taken by the agent. Let the current state of the POMDP be $s = \langle \pi_k, L, \hat{\mathbb{E}}_{\text{next}}, \hat{\mathbb{E}}_{\text{given}}, u_t \rangle$. If the agent's action, denoted as a , belongs to the set $\mathbb{E}$, then the next state of the POMDP is simply:

$$\langle \pi_k, L, \hat{\mathbb{E}}_{\text{next}} \cup a, \hat{\mathbb{E}}_{\text{given}}, u_t \rangle$$

In this case, the explanatory message a is added to the set of messages to be given in the next interaction step. In contrast, when the agent's action is $a = \text{explain}$, the interaction moves to its next step. In this scenario, the user is presented with the prefix π_{k+1} (i.e., the behavior trace up to the transition τ_{k+1}) and is given the messages in $\hat{\mathbb{E}}_{\text{next}}$. The next state of the POMDP is determined by the sequence of labels that the user of type t will assign to π_{k+1} . Consequently, the next set of possible states would have the following characteristics:

- Prefix set to π_{k+1} .
- $\hat{\mathbb{E}}_{\text{given}}$ set to $\hat{\mathbb{E}}_{\text{given}} \cup \hat{\mathbb{E}}_{\text{next}}$.

- $\hat{\mathbb{E}}_{\text{next}}$ set to $\emptyset$.
- Sequence of explicability labels, denoted as L , as given by the user.

Importantly, the user’s type remains unchanged in this transition. Assuming that each transition is independently labeled by the user, the transition probabilities for each state are computed as:

$$\prod_{i=1}^{k+1} P_{\mathcal{L}_t}(l_i | \tau_i, \hat{\mathbb{E}}_{\text{given}})$$

Here, l_i represents the i^{th} label in L .

- **$R_{\mathcal{E}}$** : The reward that the robot obtains will encompass both the cost associated with presenting an inexplicable transition and the cost of communicating an explanatory message. For a transition $\langle s, a, s' \rangle$ where s' includes π_k as the prefix and l_i represents the label assigned to transition t_i in π_k , the reward function is defined as follows:

$$R_{\mathcal{E}} = \begin{cases} -1. \lambda. C_e & \text{if } a \in \mathbb{E} \\ -1. \sum_{i=1}^k C_{\text{inexp}}. (1 - l_i) & \text{if } a = \text{explain} \end{cases}$$

- **$\Omega_{\mathcal{E}}$** : The set of all observations would be all possible explicability labels for all prefixes π_k in behavior trace being explained to the user. In addition, $\Omega_{\mathcal{E}}$ will contain a special observation denoted as l_{ϕ} .
- **$O_{\mathcal{E}}$** : The observation model is deterministic. If the interaction moves to its next step, then the observation is the label sequence L associated with the next state. Otherwise, it is simply l_{ϕ} .
- **$\gamma_{\mathcal{E}}$** : We leverage discounting to allow for on-time explanations of the form studied in Zakershahra et al. (2020).

Our formulation of the explanation reasoning into a sequential reasoning framework gives us certain advantages. For one, we are able to leverage the POMDP’s ability to naturally generate information gathering strategies when appropriate to help proactively identify the user type. In general, each observation of explicability labels, denoted as L , gives us evidence about the underlying type of the current user. Assuming that each transition is labelled independently, the belief about the current user’s type is updated as follows in each interaction step when we observe L :

$$\begin{aligned} b_{k+1}(t) &= (1/Z) \cdot P(L|\pi_{k+1}, \hat{\mathbb{E}}_{\text{given}}, t) \cdot b_k(t) \\ &= (1/Z) \cdot \prod_{i=1}^{k+1} P_{\mathcal{L}_t}(l_i|\tau_i, \hat{\mathbb{E}}_{\text{given}}) \cdot b_k(t) \end{aligned}$$

Here $b_k(t)$ denotes the probability that the current user is of type t at k^{th} interaction step, where l_i is the label provided by the user for the transition τ_i and $\hat{\mathbb{E}}_{\text{given}}$ is the set of explanatory messages that have been presented so far. To leverage the belief update, the POMDP policy could choose to employ explanation messages at a step that are designed to help reduce the possible uncertainty regarding the user’s type. For example in a case with two user types, say a given transition is inherently explicable for one type of user while it requires an explanation for the other type. By withholding any explanation, the system would be able to tell the exact user type from the next observation. Another advantage of using the sequential reasoning framework is that it allows us to leverage the fact that explanations may be non-monotonic Chakraborti *et al.* (2017), i.e. the POMDP would prevent revealing unnecessary information up front that could potentially lead to the user getting confused.

While there exist exact algorithms for solving POMDPs, they are intractable for problems with long horizon (which in the case of $\mathcal{M}_{\mathcal{E}}$ is $|\Pi| + |\mathbb{E}|$) and large observation space. Instead we will mostly focus on using approximate POMDP solvers. In particular, we will consider approximations that are myopic, i.e. their ability to account for information gathering strategies is limited and we will also be looking at

approximations that rely on at least some limited amount of lookahead.

Although reasoning about information gathering is useful, it is also one of the main complexities in POMDPs (Fern *et al.*, 2014). Thus, using myopic solvers can be highly efficient. In general, we believe that adopting a myopic approach will not substantially impact the overall interaction cost in our problem scenario. Intuitively, this can happen as a side-effect of having incorrect belief of the current user’s type. This can be understood through an example: say the robot’s current belief puts more probability to an incorrect user type and it provides an explanation tailored to that user type for a transition. Then, the actual user might still find that transition to be inexplicable. This observation would steer the belief to the correct user type.

In the following sections, we will delve into the POMDP solvers we employ for generating explanations.

4.3.3 Myopic Approximation: QMDP

QMDP (Littman *et al.*, 1995) is an efficient method for learning a policy in a POMDP. It accomplishes this by leveraging the Q-values of the underlying MDP to estimate the Q-value of a belief state. In the QMDP approach, the agent computes the Q-value for each possible state that the system might be in by using the transitions and rewards given by the underlying MDP. These Q-values are then weighted based on the current belief probability for those states. The resulting expected value serves as the approximate Q-value for the current belief state.

In our POMDP setup, let Q_t^* denote the optimal Q-value function for the MDP when the user’s type is t . Then, following QMDP, the Q-value for a belief state b and an action a can be approximated as

$$Q(b, a) \approx \sum_t b(t) \cdot Q_t^*(s_t, a). \quad (4.1)$$

where s_t is the current state of the POMDP with the user's type set to t . Intuitively, the Q-value is being computed for a scenario in which all the ambiguity of the user type would be resolved in the next step. Thus, the Q-value estimation doesn't account for the value provided by information gathering actions.

For our problem, we use a variant of the underlying MDP such that its optimal policy can be learned efficiently. The state in this MDP would consist of state components of $\mathcal{M}_{\mathcal{E}}$ excluding the labels sequence. We ignore the labels to prevent the state space from blowing up in size (which would otherwise be exponential in length of the trace). To retain the robot's ability to prevent inexplicable labels, we instead, modify the reward function $R_{\mathcal{E}}$ for the case when $a = \text{explain}$. In that case, for the MDP corresponding to a user type t , the reward value for transitioning into a state with prefix π_k is the expected inexplicability cost of the form

$$r(., \text{explain}, \langle \pi_k, \hat{\mathbb{E}}_{\text{next}}, \hat{\mathbb{E}}_{\text{given}}, t \rangle) = -1. \sum_{i=1}^k C_{\text{inexp}} \cdot P_{\mathcal{L}_t}(l_i = 0 | \tau_i, \hat{\mathbb{E}}_{\text{given}})$$

4.3.4 Myopic Approximation: QHR

This is a variant of a method proposed in Fern *et al.* (2014) to approximate the Q_t^* values in equation 4.1. Although ignoring the labels helps in reducing the state space, it is still exponential in the number of messages. Thus in the QHR approach, instead of computing the exact value of Q_t^* , we will compute the value of a suboptimal policy to approximate Q_t^* . The suboptimal policy assigns an action to any state assuming that the robot will not be allowed to provide explanations from the next interaction step. In this case, we let each action a be a set of all possible explanatory messages i.e. $a \in 2^{\mathbb{E}}$ and calculate the value of the suboptimal policy in closed form as follows

$$Q_t^{\text{approx}}(s_t, a) = -1 * (\lambda \cdot C_e + \sum_{j=k}^{|\Pi|} \sum_{i=1}^j P_{\mathcal{L}_t}(l_i = 0 | \tau_i, \hat{\mathbb{E}}_j \cup a) \cdot C_{\text{inexp}})$$

where k is the current step in the interaction & $|\Pi|$ is the number of steps in the plan trace. The estimate can be computed much faster than computing the entire policy for the underlying MDP. We recommend the use of this technique when the MDP’s state space is large and offline computation of the policy is restricted. Once computed, the value can be used to get the Q-value estimate $Q(b, a)$ for any belief state as shown in the QMDP technique (Equation 4.1).

4.3.5 POMCP with d -step lookahead

The POMCP algorithm Silver and Veness (2010) has been shown to achieve high performance for POMDPs with large state space. We used POMCP on $\mathcal{M}_{\mathcal{E}}$ as the approximation that allows for look ahead. In our work, we made two modifications to the original POMCP approach. First, while the original algorithm uses particle filter algorithm to approximate belief update during the search, we performed exact belief update as described in our POMDP formulation. Secondly, instead of always running the simulation until the terminal state, we only run the simulation till depth d in the search tree. If the node corresponding to the history encountered till d is not present in the tree, then the simulation proceeds with the rollout. Otherwise, the simulation is terminated and value for that belief node is computed using the QMDP technique.

4.4 Evaluation

4.4.1 Computational experiments

We validate our approach through experiments conducted in two domains: the disaster-rescue domain (section 4.1) and the four rooms domain (Sutton *et al.*, 1999). In the four rooms domain, the robot’s objective is to navigate to a goal location while

taking the shortest path within an environment with four rooms. The robot must avoid walls and locations with negative rewards.

The disaster-rescue domain includes five user types, as illustrated in Figure 4.1, that we designed. In contrast, for the four rooms domain, we vary both the number of user types and randomly select their model parameters. To evaluate our approach, we define a model of the domain for each user type as an MDP. The parameter values of these models depend on the knowledge possessed by each user type about the domain. For instance, in the disaster-rescue domain, the MDP for user of type B assigns a negative reward for the transition in which the robot moves into a location with a fence (see Table 4.1 for inaccurate model information that each user type possess). With these MDP models, we can simulate users of any type for our evaluation.

In our experiments with the four rooms domain, we vary the total number of user types between 2, 3, and 4. Consequently, we have three different settings for the domain, in contrast to the single setting of 5 user types for the other domain. In each setting for the four rooms domain, the model parameters for each user type are randomly assigned. This is accomplished by first defining a parameter set $\mathcal{M}$ that characterizes the MDP for the domain. In the case of this domain, $\mathcal{M}$ includes parameters such as goal locations, discounting factor, step cost, special locations with associated penalties, and the magnitude of these penalties. For each parameter i in $\mathcal{M}$, we associate two different values: $\mathcal{M}_i^r$ and $\mathcal{M}_i^t$. The robot’s model is instantiated by setting its MDP parameters to their corresponding $\mathcal{M}_i^r$ values. For each user type t , a subset of model parameters $\mathcal{M}_t \subset \mathcal{M}$ is selected, ensuring that the union of all selected model parameters across user types covers the entire parameter set $\mathcal{M}$. Each selected model parameter i is then assigned the value $\mathcal{M}_i^t$, while the remaining MDP parameters retain the value $\mathcal{M}_i^r$.

To collect the necessary data for learning the user types that exist for the do-

main and learning their labeling models, we employ simulated users, referred to as “observers” (as explained in Section 4.3.1). For each user type, we create three *observers*, collecting 100 and 300 data points per *observer* for the four rooms and disaster-rescue domains, respectively. For each *observer*, we generate traces where the robot starts from a random initial state and follows the optimal policy until it reaches a terminal state or reaches a maximum trace length of 10 steps. For each trace, a set of explanatory messages randomly selected from the set $\mathbb{E}$ is provided to the *observer*. The *observer*’s model is updated based on these messages, and labels are obtained for each step of the trace. A transition is labeled as explicable if the optimal policy in the updated *observer*’s model can generate that transition.

Once we’ve collected data from the *observers*, we have a set of labeled transitions for each user type. Using this data, we proceed to learn the labeling models, denoted as $\mathbb{L}_{pc}$, corresponding to each user type. These models represent the optimal labeling models that can be learned for each user type.

To learn the labeling models, we employ a decision tree classifier, with the input features consisting of transitions (where the state is defined by the robot’s grid location for both domains) and the set of explanatory messages. The output is the label provided by the *observer*. We evaluate the confidence in our labeling models, denoted as $P_{\mathcal{L}_t}$, by measuring the test set accuracy after training the classifier. Across all types of users and settings in both domains, the testing accuracies for the $\mathbb{L}_{pc}$ models were consistently greater than or equal to 0.98.

To evaluate our approach in scenarios where the explanatory system lacks prior knowledge of the user types, we apply the clustering strategy described earlier (Section 4.3.1) to the collected data for each setting to learn the setting’s user types & their corresponding labeling models. Our clustering approach achieved ideal results, correctly identifying the number of user types and grouping observers of the same

type together in both domains, except for one scenario where observers from two different types were clustered together. Upon analysis, it was determined that these two user types were assigned model parameters such that all transitions provided to their observers were inherently explicable, rendering explanations unnecessary. Hence, it was acceptable for them to be placed in the same cluster and considered as the same type. Ultimately, the labeling models learned for the identified user types in each setting achieved testing accuracies similar to the $\mathbb{L}_{pc}$ models (i.e., greater than or equal to 0.98).

Furthermore, we devised a straightforward baseline technique for comparison with POMDP solvers. In this baseline, a single labeling model is learned for all the labeled transitions collected from all observers in each setting. This model is then attributed to all users in that setting, essentially assuming that there is only a single user type. Since the labeling model for the target user is now known, generating explanations becomes straightforward, treating the problem as an MDP.

To evaluate both the baseline and the POMDP solvers, we define a regret measure, $\mathcal{R} = C(\mathcal{E}) - C(\mathcal{E})_{pc}$, where $C(\mathcal{E})$ is the cost of the explanation sequence generated by the solver, and $C(\mathcal{E})_{pc}$ is the cost of the explanation sequence generated by an oracle agent with knowledge of the user’s correct type and their labeling model, $\mathbb{L}_{pc}$.

We compute the regret value under various communication and inexplicability trade-off conditions by varying λ (see definition 4). The costs of communicating an explanatory message and presenting an inexplicable transition to the user are both set to 1 unit. For each value of λ , we generate three traces where the robot starts from a random initial state and follows the optimal policy until reaching a terminal state. We then use both our baseline and POMDP solvers to generate explanations for the traces for a user of each type in each setting across the two domains. The POMCP solver is used with a lookahead of 2 steps.

Domain	λ	Oracle's $C(\mathcal{E})_{pc}$	Regret ($\mathcal{R} = C(\mathcal{E}) - C(\mathcal{E})_{pc}$)			
			QMDP	POMCP	QHR	Baseline
Four rooms	0.5	0.27	0.46	0.59	0.62	7.11
	1.0	0.55	0.62	0.70	0.96	7.40
	1.5	0.83	0.66	0.61	1.29	7.62
	2.0	1.11	0.70	0.74	1.62	7.85
	2.5	1.31	0.74	0.75	2.03	8.14
Disaster rescue	0.5	0.86	1.06	2.0	3.46	11.93
	1.0	1.73	0.93	0.93	5.86	12.06
	1.5	2.6	1.0	1.0	6.53	12.2
	2.0	3.46	1.2	1.06	5.13	12.06
	2.5	4.33	1.4	1.25	5.36	11.73

Table 4.2: Average regret for the POMDP solvers and baseline technique along with the interaction cost, $C(\mathcal{E})_{pc}$, incurred by an oracle agent for varying value of λ .

Table 4.2 displays the average regret measure for both domains at each value of λ . The averages are computed across all plan traces and all users within each setting in the domain. The table also shows the average cost $C(\mathcal{E})_{pc}$ for the oracle. The low regret values indicate that our system can effectively identify the user type and provide the necessary explanations. Its performance is closely aligned with that of an oracle agent possessing the best possible labeling model for the target user. Moreover, all our techniques significantly outperform the baseline approach. Notably, among our solvers, both QMDP and POMCP exhibit similar performance while outperforming QHR.

4.4.2 User study

We conducted a user study to investigate the impact of providing personalized explanations in human-robot interactions, particularly when the robot is explaining a behavior trace to a human. The study followed a between-subject design in which participants received explanations from one of two techniques: *personalized* and *conformant*. This study focused on the disaster-rescue domain, featuring the user types described in Table 4.1. We utilized the domain that we ourselves designed to ensure that participants did not possess prior background knowledge about the task.

Explanation techniques

1. These explanations were generated using the QMDP solver (4.3.3) with λ set to 1, treating the participant as a user with an unknown type. The labels provided by the participant were used to update the robot’s belief about their type. For learning the labeling model of each user type, (i.e. learning the set $\mathbb{L}$ required by our POMDP solvers), we collected around a 1000 data points for each type from simulated *observers*. We assumed a perfect clustering exists & learned a labeling model for each type. The accuracies for learned models were > 0.99 .
2. In this technique, the robot generated explanations to ensure that the presented prefix would be explicable to a user of any type. We generated such explanations by finding the optimal policy for the MDP underlying the POMDP (Section 4.3.2) with a modified reward function that calculates the expected inexplicability cost of the plan prefix across all user types, t . Specifically, when $a = \textit{explain}$, the reward for the transition was computed as:

$$(-1. \sum_t \sum_{i=1}^k C_{inexp} \cdot P_{\mathcal{L}_t}(l_i | \tau_i, \mathbb{E}_{given})).$$

Tested hypothesis

We evaluated interactions between participants and the robot under the two explanation techniques across three dimensions by testing the following hypotheses:

- **Hypothesis 1:** Personalized explanations lead to shorter interaction times with the robot.
- **Hypothesis 2:** Personalized explanations lead to higher explicability of the robot’s behavior.
- **Hypothesis 3:** When given personalized explanations, users can retain the updated robot model at the end of the interaction.

Procedure

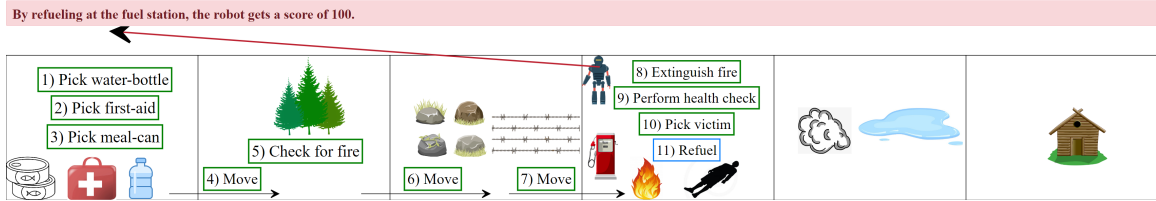


Figure 4.2: Web interface for human-robot interaction during the study, where participants provide explicability labels for each transition while receiving plan prefixes and explanations.

The user study was conducted entirely through a web-based interface while the participants were on a call with myself, with participants having the option to query

me in case they had doubts related to their task. Participants were allowed to keep their video disabled but were required to have their microphone on during the study.

Each participant was allocated one of the user types within the disaster-rescue domain and assigned to one of the explanation techniques. At the outset of the study, they received specific domain information relevant to their designated user type. Subsequently, participants underwent training for their role in the study, which revolved around evaluating the robot’s behavior.

The evaluation process involved presenting the user with a plan prefix and incrementally adding the next step in the plan at each stage of the interaction. During this process, participants were tasked with labeling each transition they encountered as either expected or unexpected. They assigned these labels based on the domain information provided and any explanations presented by the robot. Importantly, participants were encouraged to incorporate and update their domain knowledge whenever an explanation was provided during the interaction.

Once the training was complete the participant go through an eligibility test to ensure that participants accurately remembered the task description. Passing this test allowed participants to proceed to the actual interaction with the robot where they had to evaluate the robot’s behavior. We show the web interface used during the interaction in Figure 4.2.

After completing the interaction, a quiz assessed whether participants remembered the updated task description. They had to choose accurate statements from a set of four statements describing the domain. Two of these statements were inaccurate, while the other two reflected the domain parameters that would be updated by explanations that were given to the user. Each correct response earned the participant a score of 1.

Results

Measure	Conformant	Personalized	p-value
Interaction time	120.48	100.85	0.04
Inexplicable transitions	1.45	0.65	0.032
Quiz score	2.6	3.45	0.024

Table 4.3: Evaluation Measures for comparing *personalized* and *conformant* explanations

We conducted the study using between subject design with two explanation types: *personalized* and *conformant*. A total of 40 participants (aged 19 to 31, mean age = 23.9 with 10 participants being female) were involved, with each explanation type having 20 participants. The 5 user types were evenly distributed among the 20 participants for the personalized explanation.

To test each hypothesis (section 4.4.2), we gathered objective measures from the interactions. For the first hypothesis, we measured the total time participants took to complete the robot’s evaluation. For the second hypothesis, we counted the number of transitions that remained inexplicable at the end of the interaction with the robot. For the final hypothesis, we used the participants’ quiz scores. Table 4.3 provides the average values for each measure across all participants, with interaction time reported in seconds.

We conducted two-tailed t-tests and found that the difference between explanation types was statistically significant for all measures (p-values reported in Table 4.3). The results demonstrate that *personalized* explanations generated using our method outperformed *conformant* explanations across all measures.

4.5 Chapter Summary

In this chapter, we explored how a robot can tailor its explanations for its actions by taking into account the mental model of the current type of user. Importantly, the technique presented is versatile and can be applied to agents with opaque task models, as long as the set of explanatory messages initially provided to the robot describes the task dynamics in a symbolic form comprehensible to the user.

Starting with a set of explanatory messages and the robot’s task model, our explanatory system first identifies the various user types relevant to the task. Subsequently, it learns a labeling function as an approximation of their model of the task. We demonstrated how a POMDP-based formulation can leverage these labeling functions to dynamically infer the current user’s type and personalize the robot’s explanations in real-time.

We validated the effectiveness of these techniques in practical scenarios using both a disaster-rescue domain and the four-rooms domain. Our experiments covered various trade-off conditions between communication and inexplicability costs. Across the considered cases, our approaches came very close, in terms of interaction cost, to an explanatory system that already possesses knowledge of the current user’s type. Furthermore, we conducted a user study that clearly highlighted the advantages of personalizing explanations in human-robot interactions.

The work presented in this chapter was accepted as a paper in IROS 2021.

VOCABULARY EXPANSION FOR PREFERENCE SPECIFICATION IN INSCRUTABLE TASK REPRESENTATION

In this thesis, our primary focus so far has been on enhancing the interpretability of our AI agent. We aim to achieve this by generating explanations that are both easily understandable, thanks to the use of a shared symbolic vocabulary, and personalized, taking into consideration the user’s background knowledge about the task. Ensuring that our agents exhibit behavior that is comprehensible to users is crucial for their widespread adoption in our daily lives.

However, it’s equally important that the agent’s behavior aligns with human preferences. In addition to interpretability, we also aim for our AI agent to be configurable or customizable.

As demonstrated earlier in this thesis, effective communication between humans and AI agents is hindered when the agent uses a task representation that is opaque or non-interpretable to the human user. We have shown how the use of a symbolic interface or vocabulary can mitigate this problem by enabling the agent to generate explanations using a shared vocabulary. This shared vocabulary includes terms that are both understandable to humans and grounded in the agent’s internal task representation.

In this chapter, we advocate for the adoption of a similar shared vocabulary approach to allow users to specify their preferences to the agent. The users can use terms in this vocabulary to specify their preferences to the agent which can then adapt its policy to align with the preference. Consequently, symbolic interfaces serve the dual purpose of enhancing interpretability and empowering users to customize

the AI agent’s behavior to align with their preferences.

So far, we have assumed that the shared symbolic vocabulary has already been established. Our techniques in Chapter 3 did have means to understand when the existing vocabulary is insufficient to generate explanations. However, it did not have the means to add new concepts to the vocabulary (i.e., ground new symbols) when the vocabulary was insufficient. To the best of our knowledge, even other techniques that do advocate for the use of a symbolic interface (Lyu *et al.*, 2019; Illanes *et al.*, 2020; Icarte *et al.*, 2022), particularly to aid in training an agent to learn sequential decision-making tasks with fewer environmental interactions, also tend to assume that all the necessary concepts are already present in the shared vocabulary.

In contrast, in this chapter, our main focus is on giving humans the ability to add novel concepts to the shared vocabulary. This is especially important in cases where we want users to share their preferences with the agent. Since these preferences can be unique, they could likely involve concepts that are not present in the original symbolic interface.

In this work, we propose an AI system named *PRESCA* (PREference Specification through Concept Acquisition) that maintains a symbolic interface made up of concepts that the user can use to specify their preferences to the agent. If the concept relevant to the user’s preference is missing from the interface, then *PRESCA* will attempt to learn this concept online. Subsequently, the concept is also added to the interface to support preference specifications by future users. Thus, the cost of learning the concept gets amortized when future users make use of the concept. The focus of this work will be to propose a method that allows us to learn human concepts efficiently.

Currently, the primary techniques for learning user preferences involve preference-based reinforcement learning approaches (PbRL) (Christiano *et al.*, 2017; Lee *et al.*,

2021), also known as RLHF (Reinforcement Learning through Human Feedback). However, we see limitations in PbRL compared to the PRESCA framework:

- (a) For tasks where user preferences can be explicitly stated in symbolic terms, PbRL can be unnecessarily cumbersome for the user compared to *PRESCA*, which uses a symbolic vocabulary. Additionally, as we will see in section 5.2, *PRESCA* mainly relies on concept queries, where the user labels a state image as positive or negative for a given concept. This is arguably easier than answering pairwise comparison queries that are required in PbRL. In section 5.6.2, we do a user study to systematically evaluate the cognitive cost of these concept queries vs queries made for PbRL.
- (b) PbRL requires users to provide feedback for the entire task, whereas *PRESCA* focuses only on learning preference-specific concepts, reducing the volume of input required from users. In section 5.6.1, we compare the amount of feedback typically required by PbRL vs *PRESCA*.
- (c) PbRL needs to relearn preferences for each new user, while *PRESCA* adds learned concepts to the shared interface for reuse.
- (d) *PRESCA* leverages pretrained vision language models (VLMs) to bootstrap concept learning, reducing the user feedback needed. This is not possible to do as trivially with PbRL approaches.

Now to learn a novel concept, a straightforward way is to establish a grounding from the concept to system states. As we saw in chapter 3, one could learn these groundings from a set of positive and negative examples of the concept and then subsequently use a classifier to ground the concept. Obtaining these examples is a challenging problem as there is no clear way for the user to generate these examples.

One approach to obtain these examples is for the system to present the user with states and ask them whether the concept is present in each state. A naive way to generate these queries could involve randomly sampling states from the environment and asking users for their labels. However, if the positive examples are sparse in the state space, then this strategy would lead to a potentially large number of queries before sufficient positive examples are collected.

Our goal in this work is to optimize the data collection process for learning new concepts by making it more feedback-efficient. We accomplish this by automatically identifying probable positive and negative examples of the concept through the application of certain heuristics. These potential examples are then presented to the user for labeling, allowing us to acquire the necessary positive examples of the concept while significantly reducing the number of queries as would be required by random sampling.

In our approach, we employ three distinct heuristics to obtain likely positive examples of a concept. The first heuristic is causality-based, where we leverage the causal associations the target concept has with concepts already present in the symbolic interface. The second heuristic is based on the concept’s spatial locality within an image, positing that the region most pertinent to a concept is typically found within a localized patch in the image observation of the state. The third and final heuristic depends on concept proximity, operating under the assumption that states neighboring a positive example are also likely to be positive.

In Figure 5.1, we provide an overview of the user’s interaction with the PRESCA system. In the following section, we introduce the planning domain used to illustrate and evaluate our approach. This is followed by a formal description of the environment’s model and the symbolic interface employed by our AI system. We then present a methodology for efficiently learning a new concept and utilizing the learned con-

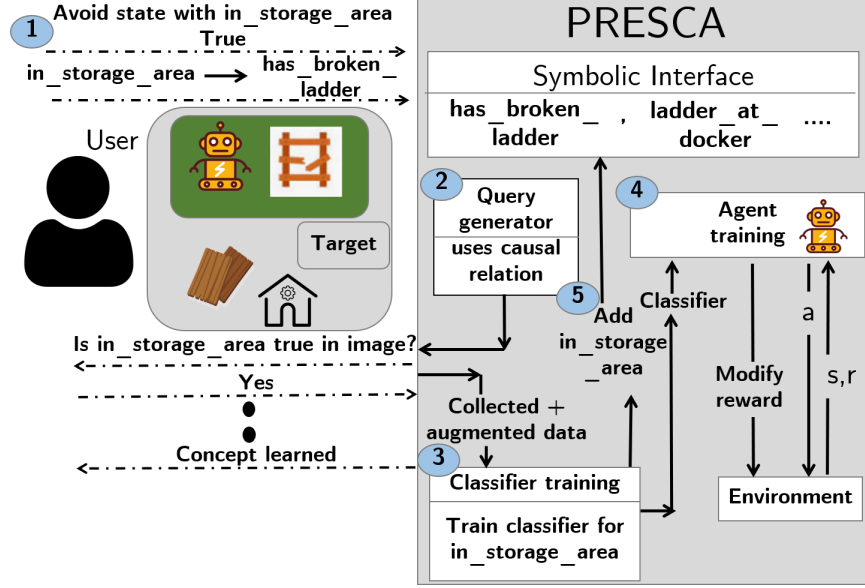


Figure 5.1: Overview of *PRESICA*. (1) The user specifies their preference in terms of some symbolic concept. If the concept is not present in the symbolic interface, then the user provides its causal relationship to some known concept. (2) *PRESICA* then generates likely positive examples and negative examples of the concept and queries their label to the user. (3) After getting the labels, *PRESICA* learns a classifier for the target concept and (4) incorporates user’s preference in agent’s training. (5) Finally, the concept is added to the interface.

cept to guide the agent’s training. In the evaluation section, we start by evaluating the reduction in feedback obtained by various components of our approach. This is followed with evaluation against PbRL using both computational experiments and a user study.

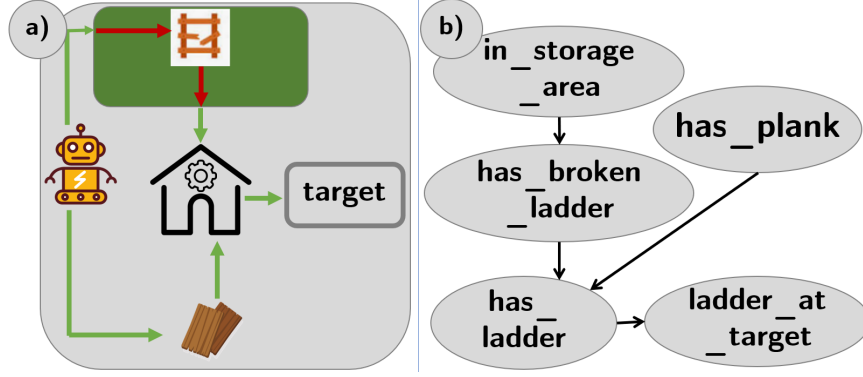


Figure 5.2: (a) Instance of the Minecraft environment with two possible plans marked with arrows. The user prefers that the agent avoid going into the storage area (indicated by the red arrows) (b) the causal model of the Minecraft environment.

5.1 Illustrative Example

We will utilize a variant of the 2-D Minecraft environment introduced by Andreas *et al.* (2017) (Figure 5.2(a)) to illustrate the concepts discussed in this paper and evaluate our proposed technique. In this environment, the primary objective is to place a ladder at a specific target location. The available agent actions encompass turning left or right, moving forward, picking up an object, executing a crafting action, taking a no-op action, and dropping the ladder when the agent reaches the target. To achieve the goal, the agent must first acquire a ladder. There are two distinct methods to accomplish this task. The agent can either pick up a plank and then use a crafting station to craft a ladder, or it can enter the storage area (highlighted in green in Figure 5.2(a)), pick up a broken ladder, and subsequently use the crafting station to repair it. Once the agent possesses the ladder, it can proceed to the target location and place it. Both of these possible plans are depicted in Figure 5.2(a).

Additionally, there is a human observer who has specific preferences regarding how the agent should complete the task. The observer’s preference is for the agent

to avoid entering the storage area. Since one of the task-solving approaches involves entering the storage area, the human needs to communicate this preference to the agent. In this scenario, the agent operates based on a state encoding that the human cannot directly interpret in terms of specific state features. However, if the human can communicate with the AI system using symbolic terms, they can simply instruct the system to avoid states in which the proposition `in_storage_area` might be true. In this context, `in_storage_area` is a propositional fact that holds true in every state where the agent is located inside the storage area.

To facilitate such a symbolic specification, it is crucial for the AI system to be capable of correctly grounding and interpreting the potential concepts that the user wishes to use. In this particular case, *PRESCA* learns a classifier that can determine whether the agent is inside the storage area in a given state. Training this classifier requires the user to provide both positive and negative examples of the `in_storage_area` concept. In this work, we aim to optimize the efficiency of this data collection process by minimizing the number of queries presented to the user for getting the labels. As mentioned earlier in this chapter, one approach to achieve this is by collecting probable positive examples of the concept and then querying the user for their labels.

To identify these likely positive examples, we employ a heuristic based on the precedence relation. Figure 5.2(b) illustrates the causal relationships between various concepts in the environment. For instance, the concept `in_storage_area` precedes the concept `has_broken_ladder`. This causal relationship implies that the agent must be inside the storage area to acquire the broken ladder. Consequently, if we have a classifier for `has_broken_ladder`, we can identify agent trajectories that conclude with the agent possessing the broken ladder. By following the causal relation, we can deduce that the state just before the agent obtains the broken ladder is a likely positive example of `in_storage_area`. In the upcoming sections, we will delve into other

heuristics applied in our work and tackle scenarios where we possess the classifier for a descendant of the target concept, albeit not necessarily its immediate child.

5.2 Problem Setting

Our problem setting involves an agent that is solving an RL problem modeled as an unknown Markov Decision Process (MDP) as defined in Sutton and Barto (2018) denoted as $\mathcal{M}$. Our setting additionally considers the agent to be goal-directed. This means that there is a set of goal states $\mathbb{G}$ and the agent tries to reach one of them.

This work focuses on human-AI interactions, where the agent engages with multiple users over time. In each interaction, a human-in-the-loop requires the agent to complete a task in accordance with their preferences. To facilitate this, we introduce the system *PRESCA* that allows users to communicate their preferences to the agent.

The challenge we encounter here pertains to the potential inscrutability of the state representation employed within the model $\mathcal{M}$. Our primary concern centers around scenarios where the model relies on an image-based state representation, making it impractical for users to express their preferences at the pixel level. To tackle this issue, we introduce a symbolic interface, akin to the shared vocabulary employed for generating explanations (as discussed in Chapter 3). This symbolic interface, denoted as F_S , consists of propositional state variables or concepts that users can interpret. In a given state s , each concept $C_z \in F_S$ can either be considered *true* or *false*.

PRESCA begins with a partial vocabulary, with some concepts already present in F_S and their accurate grounding available. This is akin to other “neuro-symbolic” approaches that assume the existence of known concepts (Lyu *et al.*, 2019; Illanes *et al.*, 2020; Icarte *et al.*, 2022). However, in specific interactions with users, if their preferences cannot be expressed using existing concepts in F_S , *PRESCA* supports learning the target concept relevant to their preference. Hereafter, we denote this

target concept as $C_{\mathbb{T}}$. Learning $C_{\mathbb{T}}$ involves acquiring a grounding in the agent’s state representation, taking the form of a binary classifier.

To learn the classifier for the target concept $C_{\mathbb{T}}$, the system collects positive and negative examples from the user. Specifically, the user is presented with state queries, where they indicate whether the concept is present or absent in the state. Once the classifier $C_{\mathbb{T}}$ is learned, it is used to train an agent’s policy that aligns with the user’s preference. Furthermore, the classifier $C_{\mathbb{T}}$ is added to the set F_S to enable preference specifications by future users.

Given this discussion, the objective of the *PRESCA* system in a single interaction with a user, when a new concept needs to be learned, is precisely defined as follows:

Given the environment $\mathcal{M}$, a user, a symbolic interface F_S , and a target concept $C_{\mathbb{T}}$, the system must: (a) minimize the number of queries made to the user to learn $C_{\mathbb{T}}$, and (b) use $C_{\mathbb{T}}$ to train an agent’s policy that achieves the goal while aligning with the user’s preference.

5.3 Assumptions for Heuristic Label Estimation

In our setup, we consider the setting where the concept $C_{\mathbb{T}}$ is relatively rare in the state space. Thus querying labels for states that have been randomly sampled from the state space, would lead to significant number of queries before enough positive labels have been collected.

Thus, to optimize for the number of label queries made to the user, we collect likely positive examples of the $C_{\mathbb{T}}$. We will now discuss specific assumptions we make in our setting that allow us to get candidate examples of concepts.

5.3.1 Causal relationship with a known concept

The *PRESCA* system assumes that we have been given a causal relations between the target concept $C_{\mathbb{T}}$ and an already known concept $C_{\mathbb{K}} \in F_S$. We go into the exact detail on how we use this causal relation to get likely examples of $C_{\mathbb{T}}$ in section 5.4.4. In this section, we give the semantics of the causal model that is used in our work.

This causal relationship is represented as a partial causal model of the domain. At a high level, the causal model for a domain is a directed graph where nodes represent concepts, and edges indicate causal associations. Figure 5.2(b) provides an example of a causal model for the Minecraft domain. In this model, connections between a concept $C_{\mathbb{Z}}$ and its parent concepts reflect abstract relationships that parent concepts have with $C_{\mathbb{Z}}$ in the domain’s dynamics.

Informally, the relation $C_{\mathbb{T}} \rightarrow C_{\mathbb{K}}$, where $C_{\mathbb{T}}$ is the only parent of $C_{\mathbb{K}}$, implies that if, in a transition $\langle s, a, s' \rangle$, the concept $C_{\mathbb{K}}$ is *true* in s' while it was *false* in s , then $C_{\mathbb{T}}$ must have been *true* in the state s . In other words, for $C_{\mathbb{K}}$ to be *true* in the next state, $C_{\mathbb{T}}$ must be *true* in the current state. This can be understood when one compares this to preconditions and effects in symbolic models like PDDL (Aeronautiques *et al.*, 1998), where the relation $C_{\mathbb{T}} \rightarrow C_{\mathbb{K}}$ may be due to some symbolic action with a precondition of $C_{\mathbb{T}}$ and an effect of $C_{\mathbb{K}}$.

The causal model itself is represented using a syntax similar to structured causal models (Glymour *et al.*, 2016). Formally, the causal model $\mathcal{M}_S$ is defined as a tuple $\langle V_S, E_S, f_S \rangle$, where V_S and E_S correspond to nodes and edges of a directed graph, with each node $v \in V_S$ represent a concept. The set f_S comprises of structural equations specifying the abstract relationship between a child concept and its parent concepts. We denote a causal relationship between multiple parent concepts $C_{\mathbb{X}_1}, C_{\mathbb{X}_2} \dots C_{\mathbb{X}_n}$ to their child concept $C_{\mathbb{Y}}$ as $\{C_{\mathbb{X}_1}, C_{\mathbb{X}_2} \dots C_{\mathbb{X}_n}\} \rightarrow C_{\mathbb{Y}}$. Each equation in f_S relates

parent concepts $C_{\mathbb{X}_1}, C_{\mathbb{X}_2}, \dots, C_{\mathbb{X}_n}$ to their common child concept $C_{\mathbb{Y}}$ as follows: $C_{\mathbb{Y}} = \mathcal{V}(C_{\mathbb{X}_1}, C_{\mathbb{X}_2}, \dots, C_{\mathbb{X}_n})$, where $\mathcal{V}$ is a Boolean function. We limit $\mathcal{V}$ to disjunctions of positive literals, but it can be extended to more complex formulas. We denote the Boolean function $\mathcal{V}$ in the structural formulae of a concept $C_{\mathbb{Y}}$ as $\mathcal{V}_{C_{\mathbb{Y}}}$.

Now, for any concept $C_{\mathbb{Z}}$, let $C_{\mathbb{Z}}(s)$ denote the predicate indicating whether $C_{\mathbb{Z}}$ is *true* in the state s . Given this notation, we define the grounding of each structural equation in transitions within the domain.

Definition 5. *Given a concept $C_{\mathbb{Y}} \in V_S$, its structural equation $C_{\mathbb{Y}} = \mathcal{V}(C_{\mathbb{X}_1}, C_{\mathbb{X}_2}, \dots, C_{\mathbb{X}_n})$, and a transition $\tau = \langle s, a, s' \rangle$ in $\mathcal{M}$, if $C_{\mathbb{Y}}(s')$ is true and $C_{\mathbb{Y}}(s)$ is false, then the grounding of the structural equation is defined as the following equation that holds for τ : $C_{\mathbb{Y}}(s') = \mathcal{V}(C_{\mathbb{X}_1}(s), C_{\mathbb{X}_2}(s), \dots, C_{\mathbb{X}_n}(s))$.*

For example, in the Minecraft domain (Figure 5.2), the structural equation for `has_ladder` is of the form $C_{has_ladder} = C_{has_broken_ladder} \vee C_{has_planks}$. Thus, if there is a transition $\tau = \langle s, a, s' \rangle$ where $C_{has_ladder}(s) \equiv \text{False}$ and $C_{has_ladder}(s') \equiv \text{True}$, then from Definition 1, we have $C_{has_broken_ladder}(s') \vee C_{has_planks}(s') \equiv \text{True}$, indicating that the agent either had the broken ladder or had planks in the original state s .

5.3.2 Concept proximity and rarity

We introduce a *concept locality* assumption: if a concept $C_{\mathbb{Z}}$ is *true* in a particular state s , then it is likely to be *true* in a nearby neighborhood of states around s . To be more specific, if we consider the set of all states S' where each state $s' \in S'$ can be reached from s within a small number of actions, then it's probable that the concept $C_{\mathbb{Z}}$ is also *true* in those states within S' . This assumption is based on the idea that the agent would typically need to execute a specific sequence of actions to make the concept no longer *true*.

Additionally, we make an assumption regarding the *rarity* of concepts. In other words, for any concept C_Z , the proportion of states in which the concept is *true* is expected to be low, denoted as $|S_{C_Z}|/|S| \ll 1$, where S_{C_Z} represents the set of states where C_Z is *true* and $|S|$ is the size of state space.

5.3.3 Concept image locality



Figure 5.3: Annotated patch corresponding to the concept in storage area

Our last assumption is based on the idea that for any positive example state of any concept C_X , there exists a localized region within the image corresponding to that state that indicates the presence of the concept. To illustrate this assumption, consider Fig 5.3, where the red bounding box highlights a specific region in the image, signifying that the robot is in the storage area. This assumption implies that there are distinctive visual cues or patterns localized within the state image that can be associated with the presence of a concept in that state.

5.4 Adaptive Supervised Learning to Learn the Target Concept

Now, we proceed to describe our approach for learning the target concept classifier C_T and collecting labeled data for it. Our data collection strategy involves obtaining likely positive and negative examples of the concept C_T . To collect likely positive examples of C_T , we rely on a causal relationship between the target concept C_T and

some known concept $C_{\mathbb{K}} \in F_S$. This causal relationship can either be provided by the user or derived from a symbolic domain model using methods such as Helmert (2004), if the domain model is available.

The causal relationship is essentially a partial specification of the causal model $\mathcal{M}_S$ (see Section 5.3.1) of the domain. We denote this causal relationship as $\mathcal{M}_{\mathbb{T} \rightarrow \mathbb{K}}$, where $\mathcal{M}_{\mathbb{T} \rightarrow \mathbb{K}} = \langle V_{\mathbb{T} \rightarrow \mathbb{K}}, E_{\mathbb{T} \rightarrow \mathbb{K}}, f_{\mathbb{T} \rightarrow \mathbb{K}} \rangle$. Here, $\langle V_{\mathbb{T} \rightarrow \mathbb{K}}, E_{\mathbb{T} \rightarrow \mathbb{K}} \rangle$ forms a subgraph of $\langle V_S, E_S \rangle$, and $f_{\mathbb{T} \rightarrow \mathbb{K}} \subseteq f_S$. This causal relation $\mathcal{M}_{\mathbb{T} \rightarrow \mathbb{K}}$ must include a path from $C_{\mathbb{T}}$ to some known concept $C_{\mathbb{K}}$. We represent this path as $\mathcal{P}_{\mathbb{T} \rightarrow \mathbb{K}}$. Additionally, it should also encompass the structural equation of the node $C_{\mathbb{K}}$ and each intermediate node along the path $\mathcal{P}_{\mathbb{T} \rightarrow \mathbb{K}}$.

Before delving into our adaptive supervised learning approach that train the classifier $C_{\mathbb{T}}$ with minimal queries made to the user, we first describe two distinct methods for collecting labeled data to train a classifier $C_{\mathbb{X}}$ when we are given a causal relation $\{C_{\mathbb{X}_1}, C_{\mathbb{X}_2} \dots C_{\mathbb{X}_n}\} \rightarrow C_{\mathbb{Y}}$ where $C_{\mathbb{X}} \in \{C_{\mathbb{X}_1}, C_{\mathbb{X}_2} \dots C_{\mathbb{X}_n}\}$ and the grounding (classifier) of $C_{\mathbb{Y}}$ is known.

5.4.1 Labeling scheme for fully supervised learning to train concept classifier

Getting seed examples of $C_{\mathbb{X}}$

We initiate this labeling scheme by collecting instances of $C_{\mathbb{X}}$ by leveraging causal relationship $\{C_{\mathbb{X}_1}, C_{\mathbb{X}_2} \dots C_{\mathbb{X}_n}\} \rightarrow C_{\mathbb{Y}}$. We label these collected instances as *seed* examples and designate the entire set as $Seed_{C_{\mathbb{X}}}$. To assemble $Seed_{C_{\mathbb{X}}}$, we start with an exploratory policy within the environment, gathering transitions in the form $\langle s, a, s' \rangle$ where $C_{\mathbb{Y}}(s')$ is *true* while $C_{\mathbb{Y}}(s)$ is *false*. According to definition 5, this implies $C_{\mathbb{X}_1}(s) \vee C_{\mathbb{X}_2}(s) \dots \vee C_{\mathbb{X}_n}(s) \equiv \text{True}$. Thus we query the user if $C_{\mathbb{X}}$ is true in this state. We make the assumption that the likelihood of $C_{\mathbb{X}}$ being the causal factor

for C_Y is higher than the probability of randomly encountering C_X in a state (based on the rarity of the concept). Therefore, gathering seed examples based on the causal model is likely to be more efficient than randomly selecting states and querying their labels.

Making informed queries

Having acquired the seed examples of the concept C_X , we proceed to make informed queries to expand the dataset for C_X . To achieve this, we capitalize on the *concept proximity* assumption (section 5.3.2) to gather probable positive examples of C_X . Our approach involves performing short random walks starting from randomly selected states within $Seed_{C_X}$. These states are then presented to the user for labeling, and this process is repeated until we collect N^+ positive examples.

In line with the *concept rarity* assumption (section 5.3.2), when we randomly sample a state from the entire state space, it is more likely to be a negative example. Therefore we opt for random state selection for candidate queries for negative examples. To be precise, we begin by gathering a large set of states denoted as $\hat{\mathcal{S}}$. We achieve this by allowing the agent to explore the environment with a uniform action selection policy over a lengthy episode. Subsequently, we draw random samples from $\hat{\mathcal{S}}$ and request labels from the user until we have accumulated $\mathcal{N}^-$ negative examples.

Upon completing the data collection process, we proceed to train a binary classifier for the concept C_X . We denote the classifier obtained through informed queries for any concept C_X as C_X^* .

5.4.2 Labeling scheme for weak supervised learning

In this approach, we leverage *concept image locality* (section 5.3.3) which allows us to automatically label positive and negative examples of C_X and train the classifier

with weak supervision.

In particular, we request the user to provide a minimum bounding box that encompasses the region indicating the presence of the concept. With multiple such relevant patches for the concept, we expand our dataset by sampling states from the state space and assigning them a positive label if they contain a patch similar to any of the patches provided by the user. Otherwise, they are assigned a negative label. To accomplish this, we need a way to measure whether a patch P is contained within a given image $\mathcal{I}$, a task commonly referred to as template matching. We calculate this as a score denoted as $Sim(P, \mathcal{I})$, which represents the maximum similarity between P and any patch in $\mathcal{I}$. The similarity between two patches is determined by the distance between their autoencoder-based embedding. Below, we go into a more comprehensive explanation of this labeling process.

The process starts by obtaining seed examples of concept $C_{\mathbf{x}}$, collected using the technique outlined in Section 5.4.1. From $Seed_{C_{\mathbf{x}}}$, we randomly select $\mathcal{W}$ states and request the user to annotate each sampled state with a patch that best indicates the presence of the concept $C_{\mathbf{x}}$. Hereafter, we call such queries as patch queries.

Let $Patch_{C_{\mathbf{x}}}$ represent the set of patches provided by the user, and let S_{aug} be a large set of state images randomly sampled from the state space. For each unique patch size k among all the patches $\mathcal{P} \in Patch_{C_{\mathbf{x}}}$, we extract patches of size k from each image in S_{aug} . We then train one autoencoder Φ on all the extracted patches.

Now, for any input patch P , let $\Phi(P)$ denote the embedding produced as the output of the encoder in Φ . Given two patches P_1 and P_2 , we use the cosine similarity between $\Phi(P_1)$ and $\Phi(P_2)$ as the similarity measure between the patches. We can now define the score $Sim(P, \mathcal{I})$ between a patch P and an image $\mathcal{I}$ that indicates if the patch P is contained in $\mathcal{I}$. Let k_P denote the size of patch P . Then, $Sim(P, \mathcal{I})$ is defined as the maximum similarity between patch P and any patch, $\mathcal{I}_P$, of size k_P in

$\mathcal{I}$. Formally, the similarity score is calculated as follows:

$$Sim(P, \mathcal{I}) = \max_{\mathcal{I}_P} \Phi(P) \cdot \Phi(\mathcal{I}_P). \quad (5.1)$$

Now, for each patch $\mathcal{P} \in Patch_{C_{\mathbf{x}}}$, we need to identify images in S_{aug} that contain a patch similar to $\mathcal{P}$. To achieve this, we rank each image $\mathcal{I} \in S_{aug}$ based on $Sim(\mathcal{P}, \mathcal{I})$. We then select the top M^+ images as those with similar patches. This process is repeated for each patch in $Patch_{C_{\mathbf{x}}}$, and the union of these selected images forms the final set of positive examples in the labeled data.

On the other hand, for negative examples, we aim to identify images in S_{aug} that do not contain patches matching any patches in $Patch_{C_{\mathbf{x}}}$. To obtain this set, we rank each image $\mathcal{I} \in S_{aug}$ based on an aggregate measure, $\overline{Sim(\mathcal{P}, \mathcal{I})}$, computed as the average of $Sim(\mathcal{P}, \mathcal{I})$ values across all $\mathcal{P} \in Patch_{C_{\mathbf{x}}}$. We then select the M^- least similar images as the set of negative examples for the dataset with synthetic labels.

Given the assumption of *concept rarity* (Section 5.3.2), we choose to keep M^+ as a small value and set M^- to be slightly larger. Once we have created this labeled dataset, we utilize it to train the classifier for $C_{\mathbf{x}}$. We denote the classifier learned using this approach for any concept $C_{\mathbf{x}}$ as $\hat{C}_{\mathbf{x}}$.

The initial $\hat{C}_{\mathbf{x}}$ is trained completely on examples with estimated labels. As we will see in our final approach, 5.4.4, we subsequently attempt to improve this classifier's accuracy by finetuning it on examples with labels taken from the user. These examples are chosen using uncertainty sampling (Lewis (1995)) To quantify the uncertainty for any given example, we use the entropy of its predicted class's distribution.

$$H(s) = -(p \log_2(p) + (1 - p) \log_2(1 - p)) \quad (5.2)$$

where $H(s)$ is the entropy of a state image s , and p is the probability of the concept being true in that image. Ideally, this would quickly increase the accuracy of $\hat{C}_{\mathbf{x}}$

which might have a poor decision boundary when trained solely on estimated labels. Notably, this approach wouldn't require as many examples as training $C_{\mathbb{X}}^*$ classifier as no synthetic labels are used to train $C_{\mathbb{X}}^*$.

It is important to note that the estimated labels in this approach can be noisy. Consequently, $\hat{C}_{\mathbb{X}}$ may be less accurate than $C_{\mathbb{X}}^*$, even though it requires potentially less feedback. We account for this in our final adaptive supervised learning approach to train our original target concept $C_{\mathbb{T}}$ as detailed in the next section.

5.4.3 *Using pretrained vision language models for feedback-efficient training*

To further minimize the feedback required by the user to train a concept classifier, we bootstrap the training process by finetuning a pretrained vision language model (VLM) instead of training a classifier from scratch for the collected data. VLMs like CLIP (Radford *et al.* (2021)), BLIP (Li *et al.* (2023)), Flamingo (Alayrac *et al.* (2022)), have shown remarkable performance in zero-shot prediction of novel concepts. While, these models have shown to be useful for zero-shot prediction, we attempt to use them after finetuning it on a few examples of the target concept. This accounts for scenarios where the concept may be highly domain-specific or rare, and zero-shot prediction by the VLM may not suffice. It's crucial to note that having an accurate concept classifier is important for learning an optimal policy that aligns with the user's preference. To see this, consider scenario where there are many false negative predictions. In this case, the agent might visit states that the user wants to avoid. Similarly, false positive examples could hinder learning by blocking states that potentially reach the goal. This could result in suboptimal policies and, in extreme cases, obstruct all paths to the goal.

5.4.4 Adaptive supervised training

We are now equipped to describe our data collection approach that outputs the concept classifier for the target concept $C_{\mathbb{T}}$ given the path $\mathcal{P}_{\mathbb{T} \rightarrow \mathbb{K}}$. We first describe the simpler case when the length of the path, $|\mathcal{P}_{\mathbb{T} \rightarrow \mathbb{K}}| = 1$. In this case, we simply collect the seed examples as described in section 5.4.1, and then proceed to learn the classifier $C_{\mathbb{T}}^*$ using informed queries (section 5.4.1). Note that we require a highly accurate classifier for $C_{\mathbb{T}}$ and thus avoid noisy data labeling (section 5.4.2).

The case of $|\mathcal{P}_{\mathbb{T} \rightarrow \mathbb{K}}| > 1$ is more involved. Let the path from the concept $C_{\mathbb{T}}$ to $C_{\mathbb{K}}$ in $\mathcal{M}_{\mathbb{T} \rightarrow \mathbb{K}}$ be of the form $C_{\mathbb{T}} \rightarrow C_{\mathbb{I}_1} \rightarrow C_{\mathbb{I}_2} \dots C_{\mathbb{I}_n} \rightarrow C_{\mathbb{K}}$. We require some seed examples of $C_{\mathbb{T}}$ and then can use informed queries to train classifier of type $C_{\mathbb{T}}^*$. To do this, we need the classifier for concept $C_{\mathbb{I}_1}$. However, in order to learn the classifier for $C_{\mathbb{I}_1}$, we need the classifier for $C_{\mathbb{I}_2}$ and so on. Thus we need to learn the concept classifiers for the intermediate concepts.

In our approach we start with some total budget B_{total} for the total number of queries that can be made to the user. Of this budget, we reserve $B_{intermediate}$ for learning intermediate concept classifiers and B_{target} for learning the target concept classifier $C_{\mathbb{T}}$. Note that, we want the $C_{\mathbb{T}}$ to be highly accurate so that the learned policy will be both optimal and will align with user's preference. To enable this, we keep B_{target} to be significantly higher than $B_{intermediate}$. Finally, for each classifier, some percent of the budget is kept to gather its seed examples. While the remaining is used to make informed queries (section 5.4.1) or finetune $\hat{C}$ (section 5.4.2).

If the intermediate concept classifier's budget is found to be insufficient to get the required number, $\mathcal{U}$, of seed examples of the its parent concept, then we stop the training and rely on rerunning the training with a increased total budget. Another option could be to use the final target's budget to train intermediate concept.

However, this could lead to insufficient budget for B_{target} and hence, we disallow this option.

Now the algorithm starts by getting seed examples of $C_{\mathbb{I}_n}$ which is possible because its immediate child $C_{\mathbb{K}}$ is known. Using the $Seed_{C_{\mathbb{I}_n}}$ we first train the classifier $\hat{C}_{\mathbb{I}_n}$ (section 5.4.2). We now define a notion of *seed* query and when such a query is *successful*. Let d be all the disjuncts in $\mathcal{V}_{C_{\mathbb{I}_n}}$. Say we use $\hat{C}_{\mathbb{I}_n}$ to get a potential seed example, $\mathcal{S}$, of $C_{\mathbb{I}_{n-1}}$ by finding a transition $\langle \mathcal{S}, a, \mathcal{S}' \rangle$ where $\hat{C}_{\mathbb{I}_n}(\mathcal{S})$ is *false* and $\hat{C}_{\mathbb{I}_n}(\mathcal{S}')$ is *true* (section 5.4.1). Then a *seed* query is defined as the query where we ask the user whether any of the concept in d are *true* in $\mathcal{S}$. If it is indeed the case, then we say that the query was *successful*. The *success* rate of such queries will depend upon the accuracy of learned classifier $\hat{C}_{\mathbb{I}_n}$. Our algorithm starts by assigning some initial value to this *success* rate denoted as ϵ and operates in iterations.

In each iteration, with probability ϵ , the algorithm will try to improve the classifier $\hat{C}_{\mathbb{I}_n}$ by querying labels for a batch of example states selected via uncertainty sampling (section 5.4.2) followed by making a batch of seed queries. Or with probability $1 - \epsilon$, the algorithm will simply perform a batch of informed queries to train $C_{\mathbb{I}_n}^*$.

Algorithm 3: Algorithm to train classifier for target concept, $C_{\mathbb{T}}$

Input : $\mathcal{M}_{\mathbb{T} \rightarrow \mathbb{K}}, \mathcal{P}_{\mathbb{T} \rightarrow \mathbb{K}} = \langle C_{\mathbb{T}} \rightarrow \dots C_{\mathbb{X}} \rightarrow \dots C_{\mathbb{K}} \rangle$

Output: classifier $C_{\mathbb{T}}$

$Seed_{C_{\mathbb{T}}} \leftarrow \text{Get_all_seed}(C_{\mathbb{T}}, \mathcal{P}_{\mathbb{T} \rightarrow \mathbb{K}}) ; // \text{ see Algorithm 5}$

$C_{\mathbb{T}} \leftarrow \text{Train_classifier_}C_{\mathbb{T}}^* ; // \text{ § 5.4.1}$

return $C_{\mathbb{T}}$;

Each time a *seed* query is made, the success rate is updated using a step-size λ with a soft target update defined:

$$\epsilon = (1 - \lambda) \cdot \epsilon_{old} + \lambda \cdot \epsilon_{target} \quad (5.3)$$

Algorithm 4: Get_single_seed

Input : C_X, C_Y **Output:** potential seed example s of C_X given $C_X \rightarrow C_Y$ **while** *True* **do** $s \leftarrow \text{Env.InitialState}()$; **for** $t \leftarrow 0$ **to** *episode_length* **do** $a \leftarrow \text{UniformActionSelection}()$; $s' \leftarrow \text{Env.ExecuteAction}(a)$; **if** $C_Y(s') \wedge \neg C_Y(s)$ **then** return s // potential example of C_X **end** $s \leftarrow s'$ **end****end**

where ϵ_{old} is the value of *success* rate before the update and ϵ_{target} is the *success* rate computed over all the *seed* queries done so far. For every *successful* query, we see if $C_{\mathbb{I}_{n-1}}$ is *true* in the queried state. The algorithm terminates when either the required number of seed examples for $C_{\mathbb{I}_{n-1}}$ have been gathered or the total budget for the intermediate concept classifier has been exhausted. In the latter case, if the current value of ϵ is greater than some threshold ϵ_t , we use $\hat{C}_{\mathbb{I}_n}$ to get the set $Seed_{C_{\mathbb{I}_{n-1}}}$. Otherwise, we simply train $C_{\mathbb{I}_n}^*$ with the collected examples which is then used to get the set $Seed_{C_{\mathbb{I}_{n-1}}}$.

Thus, when a *success* rate is high, our algorithm may avoid exhausting the intermediate budget by collecting enough seed examples of the parent concept. If that's not the case, then it switches to performing informed queries, completely exhausting

the budget and learning a more reliable classifier for the intermediate concept.

Note that adaptive supervised learning can be useful even if the entire intermediate budget is exhausted. To see this, consider the case with very low budget. In this case, adaptive supervised learning will likely invest the entire budget in finetuning $\hat{C}_{\mathbb{I}_n}$ which may now be much more accurate than any potential $C_{\mathbb{I}_n}^*$ classifier due to insufficient budget. Subsequently, $\hat{C}_{\mathbb{I}_n}$ can be subsequently used to gather enough seed examples of parent’s concepts which would not be possible if we only trained $C_{\mathbb{I}_n}^*$.

Now, once $Seed_{C_{\mathbb{I}_{n-1}}}$ has been collected, the above process can be repeated till we finally collect $Seed_{C_{\mathbb{T}}}$. Subsequently the algorithm learns the classifier $C_{\mathbb{T}}^*$. We provide the complete algorithm of learning the classifier in Algorithm 3 which references subroutines in algorithm 4 & 5.

5.5 Using the Learned Concept During Training

Once we have the classifier for $C_{\mathbb{T}}$, we can use it during the agent’s training to accommodate the user’s preferences. The user can use this concept to specify preferences such as the agent should avoid states where $C_{\mathbb{T}}$ is *true*, as in our illustrative example (section 5.1), or if they want the agent to visit some subgoal state where $C_{\mathbb{T}}$ is *true* in addition to achieving the originally assigned goal.

For the case where the user wants the agent to avoid states where $C_{\mathbb{T}}$ is *true*, we simply modify the MDP $\mathcal{M}$ ’s reward function R to create a new reward function R' . In R' , the agent receives a high negative reward, denoted as $r_{\mathbb{T}}$, whenever it visits any state where $C_{\mathbb{T}}$ is *true*. The value of $r_{\mathbb{T}}$ is set to be sufficiently negative to ensure that any optimal policy will naturally steer clear of states with $C_{\mathbb{T}}$ being *true*.

To train a policy where the agent is encouraged to visit a state where $C_{\mathbb{T}}$ is *true* , in addition to achieving the original goal state, we can extend the state representation

Algorithm 5: Get_all_seed (gets required seed examples of input concept)

Input : $C_X, \mathcal{P}_{X \rightarrow \mathbb{K}}, \mathcal{U}, N^+, N^-, \epsilon, B_{intermediate}, Seed_{budget}$

Output: $Seed_{C_X}$

$C_Y \leftarrow \text{Get_child_concept_of_}C_X, Seed_{C_X} \leftarrow \{\};$

if C_Y *is known* **then**

while $|Seed_{C_X}| < \mathcal{U} \wedge Seed_{budget}$ *left* **do**

$s \leftarrow \text{Get_single_seed}(C_X, C_Y);$ // see Algorithm 4

$C_X(s) \leftarrow \text{Query_label_from_user}(s);$

if $C_X(s)$ **then** $Seed_{C_X} = Seed_{C_X} \cup s;$

end

if $|Seed_{C_X}| = 0$ **then** return “Failure. Increase budget.”;

else

// recursive call;

$Seed_{C_Y} \leftarrow \text{Get_all_seed}(C_Y, \mathcal{P}_{Y \rightarrow \mathbb{K}});$

$\hat{C}_Y \leftarrow \text{Train_classifier_}\hat{C}_Y;$ // see section 5.4.2

$\mathcal{S}_{C_Y}^+, \mathcal{S}_{C_Y}^- \leftarrow \{\}, \{\}$ // + and - examples of C_Y ;

while *True* **do**

$p \leftarrow \text{Uniform_distribution}(0, 1);$

if $p < \epsilon$ **then**

$\mathcal{S}_{C_Y}^+, \mathcal{S}_{C_Y}^- \leftarrow \text{Queries_using_uncertainty_sampling}();$ // see section § 5.4.2

$\hat{C}_Y \leftarrow \text{Train_classifier_}\hat{C}_Y;$ // see section § 5.4.2

$s \leftarrow \text{Get_single_seed}(C_X, \hat{C}_Y);$ // See Algorithm 4

$C_X(s) \leftarrow \text{seed_query}(s);$

if $C_X(s)$ **then** $Seed_{C_X} = Seed_{C_X} \cup s;$

Update ϵ using $\epsilon = (1 - \lambda) \cdot \epsilon_{old} + \lambda \cdot \epsilon_{target};$

else

$\mathcal{S}_{C_Y}^+, \mathcal{S}_{C_Y}^- \leftarrow \text{Informed_queries}();$ // see section § 5.4.1

end

if $|Seed_{C_X}| = \mathcal{U}$ **then** break;

if $B_{intermediate}$ *exhausted* **then**

if $\epsilon < \epsilon_{threshold}$ **then** $C_Y \leftarrow \text{Train_classifier}(\mathcal{S}_{C_Y}^+, \mathcal{S}_{C_Y}^-);$

if $\epsilon > \epsilon_{threshold}$ **then** $C_Y \leftarrow \hat{C}_Y;$

$Seed_{C_X} = \text{Get_all_seed}(C_X, \mathcal{P}_{X \rightarrow Y});$

end

end

end

return $Seed_{C_X}$

by adding an additional binary variable called *flag*. This *flag* variable becomes *true* once the agent visits a state where $C_{\mathbb{T}}$ is *true*. We then assign a positive reward for any transition that results in the *flag* variable becoming *true*. Importantly, once the *flag* variable becomes *true*, it must remain *true* for every subsequent transition to prevent the policy from entering cycles.

5.6 Evaluation

5.6.1 Computational experiments

The validation of *PRESCA* is conducted in the Minecraft domain, implemented using the gym-minigrid environment (Chevalier-Boisvert *et al.* (2023)). This domain consists of 10 possible initial states, each with randomized object positions, resulting in 10 unique maps for the domain. An example image observation from this domain, along with annotated objects, is shown in Figure 5.4.

In this Minecraft domain, the agent’s goal is to drop a ladder at a designated target location. The user’s preference is that the agent must avoid entering a specific storage area. Therefore, the target concept for *PRESCA* is `in_storage_area`. The reward structure in the original domain is designed so that the optimal policy would violate the user’s preference. We then apply *PRESCA* to see if it can override this reward function to achieve the goal while following the preference.

Throughout the computational experiments, to get the user feedback, a simulated human is employed, and it accurately answers queries regarding the presence of a concept in a given state. The simulated human is also used to provide the patch indicating the presence of any concept (as described in Section 5.4.2). Examples of such patches are shown in Figure 5.5.

For evaluation, we begin by conducting an ablation study with two conditions:

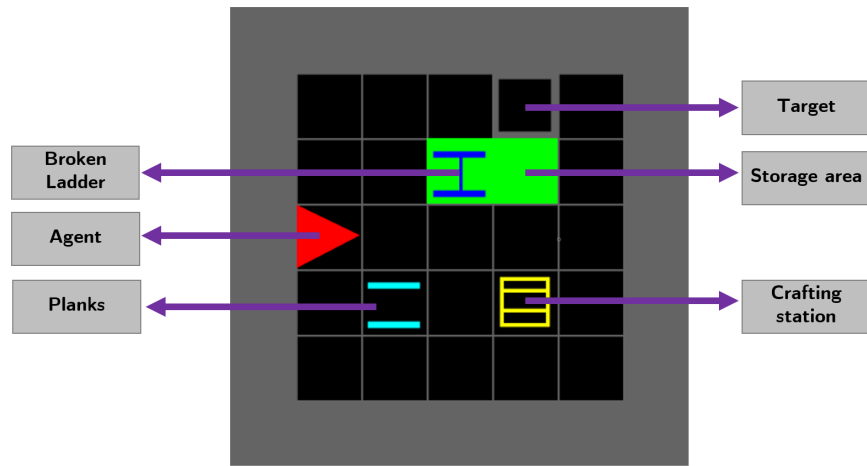


Figure 5.4: An example state image of the Minecraft domain with all objects annotated

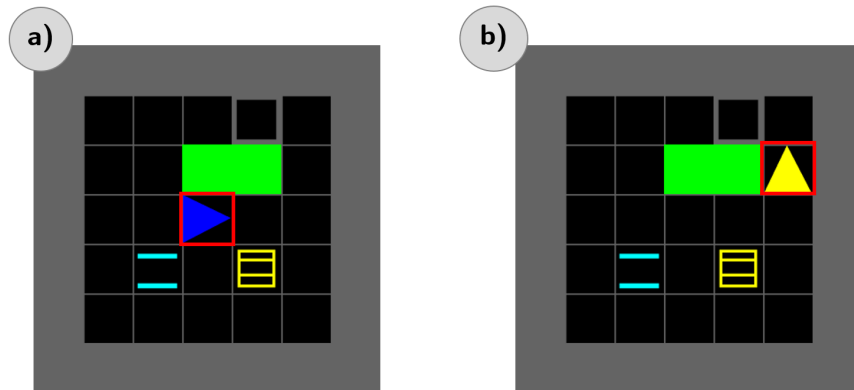


Figure 5.5: Example patches annotated with red bounding box for the concept (a) has broken ladder and (b) has ladder

employing adaptive supervised learning and utilizing a pretrained vision language model (specifically, the CLIP model Radford *et al.* (2021)) to bootstrap learning the concept classifier. We then compare PRESICA to a baseline PbRL approach Christiano *et al.* (2017) based on the amount of queries required in both techniques.

Evaluating feedback efficiency with adaptive supervised learning and VLM

We start by evaluating the reduction in the amount of feedback required to train a concept classifier when utilizing a pretrained vision language model (VLM) and adaptive supervised learning (Section 5.4.4). Therefore, we establish two conditions for this ablation study: utilizing VLM and employing adaptive supervised learning.

Each of these conditions is evaluated across three different settings, where we vary the number of intermediate concepts between `in_storage_area` and the known concept. The initially known concepts are selected from three options: `has_broken_ladder`, `has_ladder`, or `ladder_at_target`, corresponding to 0, 1, or 2 intermediate concepts, respectively (see Figure 5.2 (b)).

We will now describe each tested condition in detail. When using the pretrained vision language model for training the concept classifier, we used the CLIP (Radford *et al.* (2021)) model. Otherwise, we used a convolutional neural network with randomly initialized weights. On the other hand, for adaptive supervised learning, we trained the $\hat{C}$ classifier for intermediate concepts (section 5.4.4). When not using adaptive supervised learning, we only trained the C^* classifier for the intermediate concepts. Henceforth, we refer to the condition where we do not use adaptive learning as “non-adaptive learning”.

For each classifier, we keep the maximum budget for collecting seed examples of any given classifier to be 0.5 of the total budget for that classifier. For parameters specific to adaptive supervised learning (section 5.4.2), we used a $\mathcal{W} = 1$ patch

query for each intermediate concept classifier and the parameters M^+ and M^- as $\langle M^+ = 0.01 \cdot |S_{aug}|, M^- = 0.1 \cdot |S_{aug}| \rangle$ where S_{aug} is a large set of state images sampled from the environment by running an agent with uniform action selection. The feedback batch size, B , was kept as $1/5^{\text{th}}$ the budget for intermediate concept classifier. Finally, we keep the initial estimate of the *success* rate, $\epsilon = 1.0$ and the step-size, $\lambda = 0.2$ (section 5.4.4).

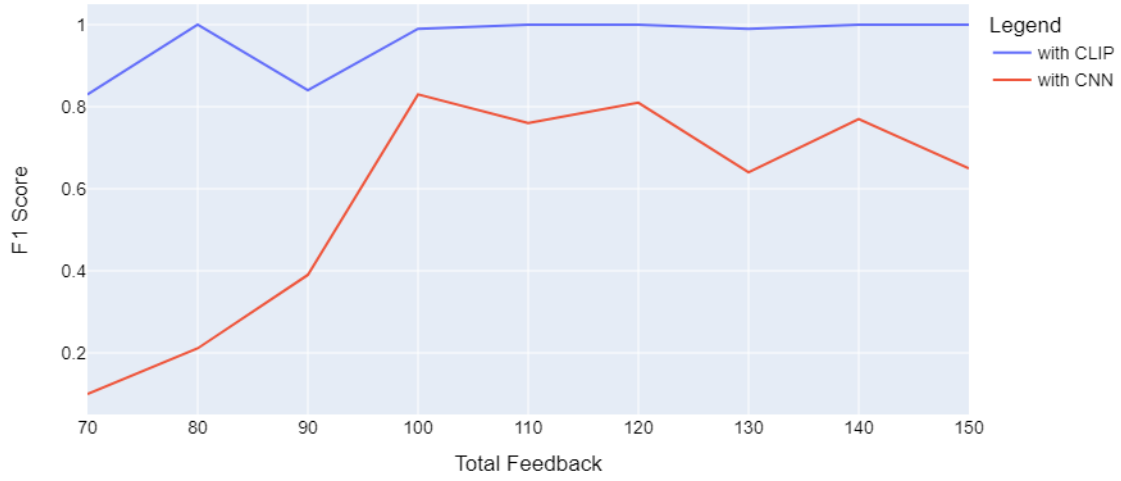


Figure 5.6: F1 score of the trained classifier for the concept `in_storage_area` against the total feedback for the case where number of intermediate concepts = 0

To evaluate each of the techniques, for the cases with number of intermediate concepts > 1 , we vary the maximum budget for the learning the intermediate concept classifiers from 10 to 50 with increments of 5, and the maximum budget for learning the concept classifier for the target concept from 130 to 170 with increments of 5. For the case when there is no intermediate concept between target concept and the known concept, we vary the budget for learning the target concept from 70 to 150 with increments of 10.

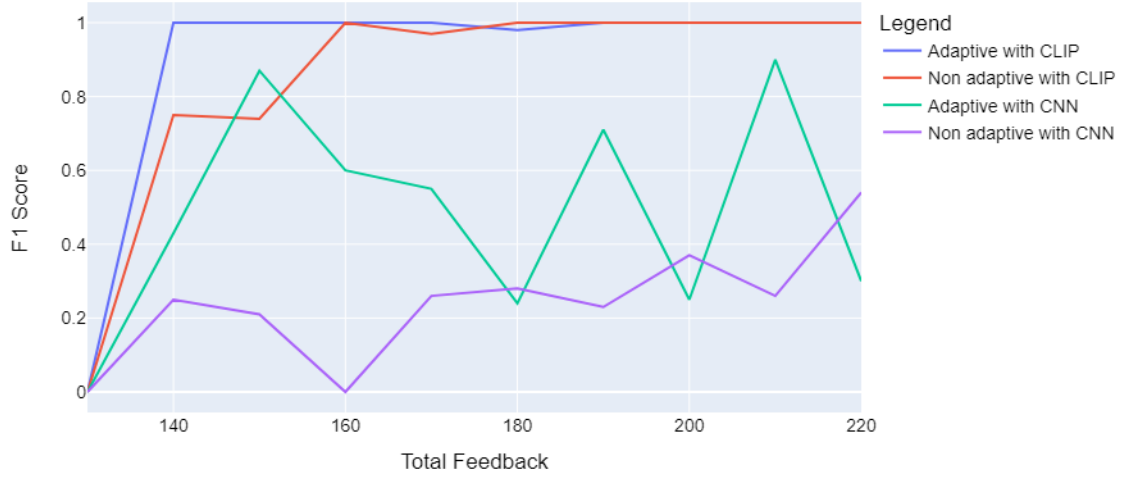


Figure 5.7: F1 score of the trained classifier for the concept `in_storage_area` against the total feedback for the case where number of intermediate concepts = 1

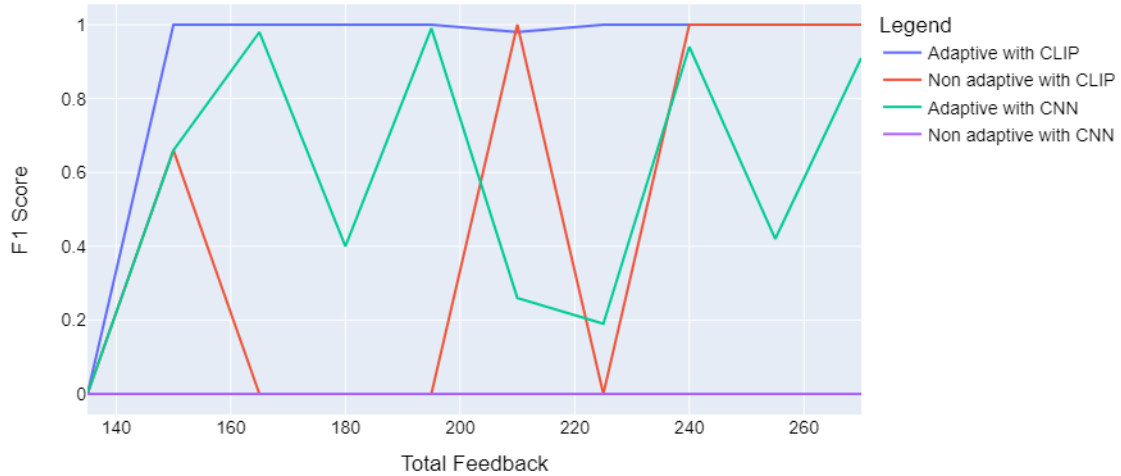


Figure 5.8: F1 score of the trained classifier for the concept `in_storage_area` against the total feedback for the case where number of intermediate concepts = 2

For each of these cases, we compute the f-1 score for the target concept by evaluating over a large set of examples with correct labels. We plot the performance of each technique varying the number of intermediate concepts between 0, 1, and 2 as displayed in Figure 5.6, 5.7 and 5.8 respectively. As we can see from the figures, using both adaptive supervised learning and a VLM, we require the least amount of feedback to train the target concept classifier. This is followed by technique where we use non-adaptive learning with VLM. Finally, the baseline technique with non-adaptive learning with CNN performs the worst among all the techniques.

Evaluation against PbRL

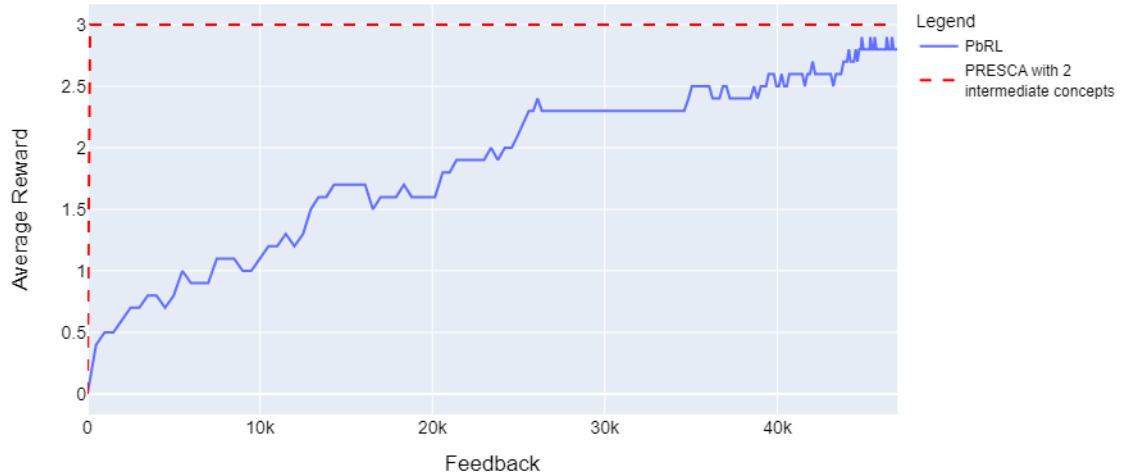


Figure 5.9: Total feedback vs average reward per map

We begin by training a reinforcement learning (RL) agent on the Minecraft domain with its dynamics modified using the concept classifiers, `in_storage_area`, which we have learned for settings with 0, 1, and 2 intermediate concept classifiers. For each setting, we select the classifier with a perfect f-1 score (1.0) that required the least amount of

feedback. This amounts to a total feedback of 80, 140, and 150 respectively for 0, 1, and 2 intermediate concepts. We then adjust the Markov Decision Process (MDP) of the domain to penalize the agent with negative rewards if it enters any state predicted as true for the concept `in_storage_area` in each setting. Upon running these trained RL agents on each map of the domain, we observe that the agents successfully achieve their goals for each setting while avoiding the storage area.

More importantly, we wanted to compare our approach to techniques like PbRL (Christiano *et al.* (2017)). We believe that PbRL would typically requires much more feedback compared to the feedback needed to learn the concept relevant to the preference. This is because PbRL relies on user-provided feedback for the entire task, not leveraging environmental rewards, unlike our method (PRESCA) which focuses solely on learning preference-specific concepts. Once these concepts are learned, we adjust the environmental reward and utilize any off-the-shelf RL approach to learn the preferred policy.

We demonstrate this empirically by applying PbRL to our Minecraft domain. To answer pairwise comparison required in PbRL, we utilize a simulated user or an oracle. We designed the oracle to provide feedback based on the agent’s proximity to achieving the goal. To measure this proximity, we define a sequence of subgoals necessary for the preferred policy to reach the goal of `ladder.at_target`, namely `has_plank`, `has_ladder`, and finally `ladder.at_target`. The oracle then selects the preferred trajectory between two given trajectories based on the following rules:

- If the next required subgoal differs in the two trajectories, the oracle prefers the one with the subgoal closer to the goal.
- If the next required subgoal is the same in both trajectories, the oracle prefers the one requiring fewer actions to achieve the subgoal.

- Finally, if the agent violates the preference in any trajectory or is holding a broken ladder, indicating a visit to the storage area, the trajectory is not preferred.

Using this oracle, we apply PbRL to learn the preferred policy. For the Minecraft domain, we experimented with two settings for training the reward model: one with a single reward model for all maps and another with separate reward models for each map. Up to a total feedback of 10,000, we observed similar performance in both cases. Subsequently, we continued training only with separate reward models as it allowed parallel training of each reward model and corresponding policy for each map. We limited the maximum number of queries made to the oracle to 10,000 for each map. In figure 5.9, we illustrate the average reward for the learned policy across the 10 maps against the total feedback obtained. The agent receives a reward of 1 for achieving each subgoal and incurs a penalty of -2 for violating the preference. Consequently, the maximum reward the agent can obtain is 3. As seen in the figure 5.9, it takes around 50,000 feedback queries for PbRL to achieve similar performance to that obtained with just 150 queries (for the most challenging setting of learning 2 intermediate concept classifiers). This highlights that expanding vocabulary can be more efficient compared to relying solely on pairwise comparison-based feedback.

5.6.2 User study

In the previous section, we compared PRESCA with a baseline PbRL technique based on the number of queries made to the user. While this comparison provides insight into the efficiency of expanding vocabulary for explicit knowledge domains, the evaluation is not entirely complete. This is because the type of queries made during PbRL are notable different than the ones made by the PRESCA system. Henceforth, we refer to the queries made by PRESCA system as *concept query* and ones made PbRL as *pairwise comparison query*.

We hypothesize that these query types incur different cognitive costs. To investigate this, we conducted a user study. The study employed a within-subject design, where each participant encountered both query types. We aimed to compare the cognitive cost of each query type using both objective and subjective measures. In the subsequent sections, we’ll describe the domain used for the study, outline our hypotheses, detail the study procedure, and present the results.

Domain details

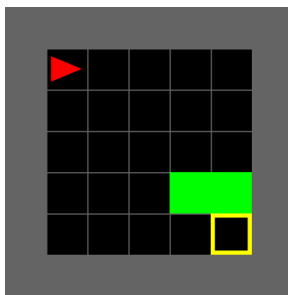


Figure 5.10: Domain used for the user study. The robot’s task is to reach the goal (cell with yellow edges), while avoiding going over the carpet area (cell marked as green square). The domain has 5 possible maps with varying locations of the goal and the carpet.

In this study, we utilized a simple grid world domain with a straightforward task for the agent (Figure 5.10). We chose this domain for its simplicity, enabling any lay user to quickly grasp the task dynamics and provide accurate answers to both concept and pairwise comparison queries. In this domain (Figure 5.10), the agent’s task is to reach a yellow square. It can execute three actions: rotate left, rotate right, and move forward to navigate the grid. Additionally, there’s a constraint that the agent must avoid going over carpet (grid cells marked as green), representing user preferences for the task.

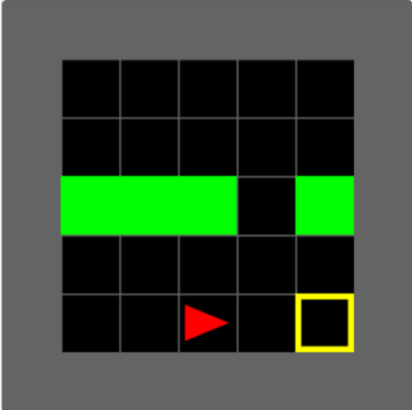
Tested Hypothesis

We evaluated the cost of making concept versus pairwise comparison queries across two dimensions by testing the following hypotheses:

- **Hypothesis 1:** Users can answer concept queries faster than pairwise comparison queries.
- **Hypothesis 2:** Pairwise comparison queries incur a higher cognitive load on the user than concept queries.

Procedure

Image



Is the agent on the carpet?

Select

Yes	☐
No	☐
Can't Say	☒

Send response

Figure 5.11: An example of a *concept query* for the concept `robot.on_carpet`

The user study was conducted entirely via a web-based interface on Prolific (Palan and Schitter (2018)). The study began with an overview explaining that participants would undertake two tasks. In both tasks, participants were tasked with guiding a robot playing a game, using different communication methods for each task. At the end of the study, participants would evaluate which method they found easier. These

In which clip does robot play better?	Select
Clip 1	☐
Clip 2	☐
Tie	☐
Can't Say	☒

Send response

Figure 5.12: An example of a *pairwise comparison* query. When clicked, the clip will play, showing the robot’s actions in the game. Participants are tasked with selecting the clip where the robot’s performance is better. They may also choose the “tie” option to indicate similar performance or select the “can’t say” option if they are unable to make the comparison.

tasks corresponded to answering *concept queries* or *pairwise comparison queries*. Participants completed the two tasks sequentially, with the order of query types randomly assigned to ensure counterbalanced results.

The overview was followed by a domain description, where we described the domain as a game that the robot is tasked with winning. They were informed that the robot wins by reaching the goal cell (marked with yellow edges) while avoiding the carpet (cells marked as green squares). Participants were shown all possible actions the agent could take, along with state images before and after each action. Additionally, participants were presented with five maps. In each map, the location of the goal and carpet was altered.

Participants then completed two tasks: one answering *concept queries* (Figure 5.11) and the other answering *pairwise comparison queries* (Figure 5.12). The order of these tasks was randomized for each participant. Below is a detailed description of

how each task was presented:

- **Answering concept queries:** Participants were tasked with determining if the concept `robot_on_carpet` was true for each query. We start by describing the task to the participant. They are shown a few positive and negative image for the concept `robot_on_carpet`. This is followed by a quiz which accessed their understanding of the task. The participants then proceeded to the main task, where they were shown ten images in sequence and they indicated the presence of the concept in each image.
- **Answering pairwise comparison queries:** In this task, participants viewed two clips of the robot playing the game and selected the trajectory that showed better performance. Additional options allowed participants to select if performance was the same or if they were unable to compare. The task began with a description where we show an example trajectory of the robot finishing the game. We additionally show an example query to the user to make them familiar with the task. This is followed by a quiz where the user is shown 3 robot's trajectories, and they must pick the one where the robot wins the game. Completing the quiz ensured that the participant understood the task's dynamics. Finally, they move to the main task, where they viewed ten pairs of clips and selected the one with better performance or indicated a tie, or their inability to judge.

For each task, we note the time taken to answer the each query to help test hypothesis 1. Once both tasks are completed, the participant is presented with a evaluation questionnaire. We adapted the NASA TLX load index (Hart (2006)) with questions that compared the two tasks based on 6 dimensions: mental demand, physical demand, temporal demand, performance, effort and frustration. Table 5.1,

shows each dimension along with the corresponding question we asked the participant. We end the study once the user has completed the questionnaire.

Results

Workload Dimension	Question	No: of participants preferring concept query (out of 50)	p-value (binomial test)
Mental demand	Which task felt less mentally demanding?	50	<0.005
Physical demand	Which task felt less physically demanding?	48	<0.005
Temporal demand	In which task did the pace felt less hurried or rushed?	35	<0.005
Frustration level	Which task made you feel less irritated, stressed and annoyed?	45	<0.005
Effort	In which task did you have to work less to achieve your performance level?	50	<0.005
Performance	Which task did you feel more successful in accomplishing?	41	<0.005

Table 5.1: Comparing the cognitive load for concept query vs pairwise comparison query. Each row represents one of the dimension used in NASA TLX workload index. We adapted the questions to let participant choose which query was better for each dimension. The participants chose concept query to be better for all the dimensions, and the preference was found to be statistically significant in each case.

We conducted a within subject study with two query types: *concept query* and *pairwise comparison query*. A total of 50 people participated in the study aged between 22 and 86 with a mean age 34.2. Among the participants, 20 were males 29 were females, while 1 refused to provide their gender.

For the first hypothesis, we measured the average time taken by the participant

to answer each of the queries. We found that on average, the participants took 14.6 seconds to answer a *pairwise comparison* query, while they only took 3.2 seconds to answer a *concept* query. We also found this difference to be statistically significant based on a two tailed t-test with p-value < 0.005 .

For the second hypothesis, we asked them questions adapted from the NASA TLX questionnaire as described in the previous section to compare the cognitive load of each query type across 6 dimensions. Table shows each of the questions, along with the number of times user picked *concept query* over *pairwise comparison query* for each of the question. As shown in the table, for all the questions, participants favored *concept query* and we additionally found the results to be statistically significant for each case using a binomial test.

5.7 Chapter Summary

In this chapter, we introduced the *PRESCA* system, which enables everyday users to convey their preferences to AI agents. *PRESCA* achieves this by maintaining a symbolic interface composed of user-understandable concepts. Crucially, it also has the capability to learn new concepts when necessary. We demonstrated how *PRESCA* leverages causal relationships between concepts, an adaptive supervised learning approach and pretrained VLMs to enhance the efficiency of the feedback process. We further evaluate *PRESCA* against PbRL technique and saw that *PRESCA* requires far less feedback than PbRL and the queries used in our approach are less cognitively taxing and take less time to answer than pairwise comparison queries that are used in PbRL. Finally, a version of this work, was presented at NeurIPS 2022 Workshop on Human in the Loop Learning.

CONCLUSION

In this thesis, we have developed mechanisms aimed at enhancing the configurability and interpretability of agents with opaque task representations. Our approach establishes a symbolic vocabulary to allow communication between humans and agents. Here, the agent utilizes vocabulary terms to generate explanations for its actions, while humans use these terms to convey their preferences to the agent. Additionally, we introduced the *PRESCA* framework, that lets users augment the vocabulary with terms specific to their preferences. Furthermore, we demonstrated how our agents can adapt explanations to individual users, ensuring that the level of detail aligns with their prior knowledge of the task.

To illustrate the effectiveness of our techniques, we conducted comprehensive evaluations. The methodologies outlined in chapters 3, 4, and 5 were evaluated using both computational experiments and human subject studies. Our findings demonstrate a significant improvement in human-AI interaction when employing a symbolic vocabulary for communication, particularly when compared to current go to approaches that rely on direct communication through pixel features and video clips of agent actions. Notably, users exhibited a preference for symbolic explanations over saliency map-based ones, and showed better understanding of the domain dynamics when given symbolic explanations. Similarly, users favored responding to simple concept queries over pairwise comparison queries prevalent in PbRL approaches.

While we believe this work to be useful in improving human-AI interaction, there are several possible directions for improvement. In particular, in this thesis we dealt exclusively with explicit knowledge tasks which allowed us to use symbolic vocabu-

lary for explanation and preference specification. However, a more general domain, would involve both explicit and tacit knowledge (example driving a car). In such cases, a hybrid approach must be designed combines symbolic interface and PbRL in one system. Furthermore, *PRESCA* in particular, was primarily designed to take capture human’s preferences. However, the same system can be extended to speed up learning in environments with sparse rewards. In this case, the user can use *PRESCA* to ground multiple subgoal concepts thus providing a more dense reward signal for speeding up RL training. Finally, our personalized explanation technique has potential to be applied to a broader range of problems, particularly in personalized teaming scenarios. In this case, we can extend the concept of user types to additionally capture preferences for specific collaboration strategies. Subsequently, with new users, we can avoid learning their model from scratch and instead map them to one of the known types. Overall, this could lead to a more efficient human-AI teaming.

REFERENCES

- Aeronautiques, C., A. Howe, C. Knoblock, I. D. McDermott, A. Ram, M. Veloso, D. Weld, D. W. SRI, A. Barrett, D. Christianson *et al.*, “Pddl— the planning domain definition language”, Technical Report, Tech. Rep. (1998).
- Alayrac, J.-B., J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds *et al.*, “Flamingo: a visual language model for few-shot learning”, *Advances in neural information processing systems* **35**, 23716–23736 (2022).
- Andreas, J., D. Klein and S. Levine, “Modular multitask reinforcement learning with policy sketches”, in “International conference on machine learning”, pp. 166–175 (PMLR, 2017).
- Arumugam, D., J. K. Lee, S. Saskin and M. L. Littman, “Deep reinforcement learning from policy-dependent human feedback”, *arXiv preprint arXiv:1902.04257* (2019).
- Atrey, A., K. Clary and D. Jensen, “Exploratory not explanatory: Counterfactual analysis of saliency maps for deep reinforcement learning”, *arXiv preprint arXiv:1912.05743* (2019).
- Bau, D., B. Zhou, A. Khosla, A. Oliva and A. Torralba, “Network dissection: Quantifying interpretability of deep visual representations”, in “Proceedings of the IEEE conference on computer vision and pattern recognition”, pp. 6541–6549 (2017).
- Brockman, G., V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba, “Openai gym”, *arXiv preprint arXiv:1606.01540* (2016).
- Bukharin, A., Y. Li, P. He, W. Chen and T. Zhao, “Deep reinforcement learning from hierarchical weak preference feedback”, *arXiv preprint arXiv:2309.02632* (2023).
- Cai, C. J., E. Reif, N. Hegde, J. Hipp, B. Kim, D. Smilkov, M. Wattenberg, F. Viégas, G. S. Corrado, M. C. Stumpe *et al.*, “Human-centered tools for coping with imperfect algorithms during medical decision-making”, in “Proceedings of the 2019 chi conference on human factors in computing systems”, pp. 1–14 (2019).
- Chakraborti, T., S. Sreedharan, Y. Zhang and S. Kambhampati, “Plan explanations as model reconciliation: Moving beyond explanation as soliloquy”, *arXiv preprint arXiv:1701.08317* (2017).
- Chevalier-Boisvert, M., B. Dai, M. Towers, R. de Lazcano, L. Willems, S. Lahlou, S. Pal, P. S. Castro and J. Terry, “Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks”, *CoRR abs/2306.13831* (2023).
- Christiano, P. F., J. Leike, T. Brown, M. Martic, S. Legg and D. Amodei, “Deep reinforcement learning from human preferences”, *Advances in neural information processing systems* **30** (2017).

- Cui, Y. and S. Niekum, “Active reward learning from critiques”, in “2018 IEEE international conference on robotics and automation (ICRA)”, pp. 6907–6914 (IEEE, 2018).
- Fern, A., S. Natarajan, K. Judah and P. Tadepalli, “A decision-theoretic model of assistance”, *Journal of Artificial Intelligence Research* **50**, 71–104 (2014).
- Freund, Y., R. Schapire and N. Abe, “A short introduction to boosting”, *Journal-Japanese Society For Artificial Intelligence* **14**, 771–780, 1612 (1999).
- Ghorbani, A., J. Wexler, J. Y. Zou and B. Kim, “Towards automatic concept-based explanations”, *Advances in neural information processing systems* **32** (2019).
- Glymour, M., J. Pearl and N. P. Jewell, *Causal inference in statistics: A primer* (John Wiley & Sons, 2016).
- Greydanus, S., A. Koul, J. Dodge and A. Fern, “Visualizing and understanding atari agents”, in “International conference on machine learning”, pp. 1792–1801 (PMLR, 2018).
- Hart, S. G., “Nasa-task load index (nasa-tlx); 20 years later”, in “Proceedings of the human factors and ergonomics society annual meeting”, vol. 50, pp. 904–908 (Sage publications Sage CA: Los Angeles, CA, 2006).
- Hayes, B. and J. A. Shah, “Improving robot controller transparency through autonomous policy explanation”, in “Proceedings of the 2017 ACM/IEEE international conference on human-robot interaction”, pp. 303–312 (2017).
- Helmert, M., “A planning heuristic based on causal graph analysis.”, in “ICAPS”, vol. 16, pp. 161–170 (2004).
- Icarte, R. T., T. Q. Klassen, R. Valenzano and S. A. McIlraith, “Reward machines: Exploiting reward function structure in reinforcement learning”, *Journal of Artificial Intelligence Research* **73**, 173–208 (2022).
- Illanes, L., X. Yan, R. T. Icarte and S. A. McIlraith, “Symbolic plans as high-level instructions for reinforcement learning”, in “Proceedings of the international conference on automated planning and scheduling”, vol. 30, pp. 540–550 (2020).
- Iyer, R., Y. Li, H. Li, M. Lewis, R. Sundar and K. Sycara, “Transparency and explanation in deep reinforcement learning neural networks”, in “Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society”, pp. 144–150 (2018).
- Kaelbling, L. P., M. L. Littman and A. R. Cassandra, “Planning and acting in partially observable stochastic domains”, *Artificial intelligence* **101**, 1-2, 99–134 (1998).
- Kim, B., M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viegas *et al.*, “Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav)”, in “International conference on machine learning”, pp. 2668–2677 (PMLR, 2018).

- Knox, W. B. and P. Stone, “Interactively shaping agents via human reinforcement: The tamer framework”, in “Proceedings of the fifth international conference on Knowledge capture”, pp. 9–16 (2009).
- Lee, K., L. Smith and P. Abbeel, “Pebble: Feedback-efficient interactive reinforcement learning via relabeling experience and unsupervised pre-training”, arXiv preprint arXiv:2106.05091 (2021).
- Lewis, D. D., “A sequential algorithm for training text classifiers: Corrigendum and additional data”, in “Acm Sigir Forum”, vol. 29, pp. 13–19 (ACM New York, NY, USA, 1995).
- Li, J., D. Li, S. Savarese and S. Hoi, “Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models”, in “International conference on machine learning”, pp. 19730–19742 (PMLR, 2023).
- Littman, M. L., A. R. Cassandra and L. P. Kaelbling, “Learning policies for partially observable environments: Scaling up”, in “Machine Learning Proceedings 1995”, pp. 362–370 (Elsevier, 1995).
- Lyu, D., F. Yang, B. Liu and S. Gustafson, “Sdrl: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning”, in “Proceedings of the AAAI Conference on Artificial Intelligence”, vol. 33, pp. 2970–2977 (2019).
- MacGlashan, J., M. K. Ho, R. Loftin, B. Peng, G. Wang, D. L. Roberts, M. E. Taylor and M. L. Littman, “Interactive learning from policy-dependent human feedback”, in “International conference on machine learning”, pp. 2285–2294 (PMLR, 2017).
- Madumal, P., T. Miller, L. Sonenberg and F. Vetere, “Explainable reinforcement learning through a causal lens”, in “Proceedings of the AAAI conference on artificial intelligence”, vol. 34, pp. 2493–2500 (2020).
- Miller, T., “Explanation in artificial intelligence: Insights from the social sciences”, *Artificial intelligence* **267**, 1–38 (2019).
- Palan, S. and C. Schitter, “Prolific. ac—a subject pool for online experiments”, *Journal of Behavioral and Experimental Finance* **17**, 22–27 (2018).
- Puri, N., S. Verma, P. Gupta, D. Kayastha, S. Deshmukh, B. Krishnamurthy and S. Singh, “Explain your move: Understanding agent actions using specific and relevant feature attribution”, arXiv preprint arXiv:1912.12191 (2019).
- Radford, A., J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, “Learning transferable visual models from natural language supervision”, in “International conference on machine learning”, pp. 8748–8763 (PMLR, 2021).
- Russell, S. and P. Norvig, “Ai a modern approach”, *Learning* **2**, 3, 4 (2005).

- Satopaa, V., J. Albrecht, D. Irwin and B. Raghavan, “Finding a” kneedle” in a haystack: Detecting knee points in system behavior”, in “2011 31st international conference on distributed computing systems workshops”, pp. 166–171 (IEEE, 2011).
- Schrader, M.-P. B., “gym-sokoban”, <https://github.com/mpSchrader/gym-sokoban> (2018).
- Selvaraju, R. R., A. Das, R. Vedantam, M. Cogswell, D. Parikh and D. Batra, “Grad-cam: Why did you say that?”, arXiv preprint arXiv:1611.07450 (2016).
- Silver, D. and J. Veness, “Monte-carlo planning in large pomdps”, *Advances in neural information processing systems* **23** (2010).
- Sreedharan, S., S. Kambhampati *et al.*, “Handling model uncertainty and multiplicity in explanations via model reconciliation”, in “Proceedings of the International Conference on Automated Planning and Scheduling”, vol. 28, pp. 518–526 (2018a).
- Sreedharan, S., A. Kulkarni and S. Kambhampati, *Explainable Human-AI Interaction: A Planning Perspective* (Springer Nature, 2022).
- Sreedharan, S., A. Olmo, A. P. Mishra and S. Kambhampati, “Model-free model reconciliation”, arXiv preprint arXiv:1903.07198 (2019).
- Sreedharan, S., S. Srivastava and S. Kambhampati, “Hierarchical expertise level modeling for user specific contrastive explanations.”, in “IJCAI”, pp. 4829–4836 (2018b).
- Sundararajan, M., A. Taly and Q. Yan, “Axiomatic attribution for deep networks”, in “International conference on machine learning”, pp. 3319–3328 (PMLR, 2017).
- Sutton, R. S. and A. G. Barto, *Reinforcement learning: An introduction* (MIT press, 2018).
- Sutton, R. S., D. Precup and S. Singh, “Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning”, *Artificial intelligence* **112**, 1-2, 181–211 (1999).
- van der Waa, J., J. van Diggelen, K. v. d. Bosch and M. Neerincx, “Contrastive explanations for reinforcement learning in terms of expected consequences”, arXiv preprint arXiv:1807.08706 (2018).
- Warnell, G., N. Waytowich, V. Lawhern and P. Stone, “Deep tamer: Interactive agent shaping in high-dimensional state spaces”, in “Proceedings of the AAAI conference on artificial intelligence”, vol. 32 (2018).
- Wikipedia contributors, “Montezuma’s revenge (video game) — Wikipedia, the free encyclopedia”, URL [https://en.wikipedia.org/w/index.php?title=Montezuma%27s_Revenge_\(video_game\)&oldid=922152932](https://en.wikipedia.org/w/index.php?title=Montezuma%27s_Revenge_(video_game)&oldid=922152932), [Online; accessed 14-January-2020] (2019).

Zahavy, T., N. Ben-Zrihem and S. Mannor, “Graying the black box: Understanding dqns”, in “International conference on machine learning”, pp. 1899–1908 (PMLR, 2016).

Zakershahrak, M., Z. Gong, N. Sadassivam and Y. Zhang, “Online explanation generation for planning tasks in human-robot teaming”, in “2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)”, pp. 6304–6310 (IEEE, 2020).

Chapter 7

ASU IRB APPROVAL



APPROVAL: EXPEDITED REVIEW

[Subbarao Kambhampati](#)
IAFSE-SCAI Computer Science and Engineering
480/965-0113
SUBBARAO.KAMBHAMPATI@asu.edu

Dear [Subbarao Kambhampati](#):

On 8/28/2023 the ASU IRB reviewed the following protocol:

Type of Review:	Initial Study
Title:	Human Factor Study on Planning for Human Robot Teaming
Investigator:	Subbarao Kambhampati
IRB ID:	STUDY00018263
Category of review:	(6) Voice, video, digital, or image recordings (7)(a) Behavioral research (7)(b) Social science methods
Funding:	Name: DOD: Navy, Grant Office ID: FP00013861, Funding Source ID: N00014-18-1-2442; Name: DOD: Navy, Grant Office ID: FP00015668, Funding Source ID: N14-18-1-2840; Name: DOD: Navy, Grant Office ID: FP00032610, Funding Source ID: N00014-23-1-2409
Grant Title:	FP00013861, FP00015668, FP00032610,
Grant ID:	FP00013861, FP00015668, FP00032610,
Documents Reviewed:	• 2409_ONR-Trust-Proposal-Kambhampati.pdf, Category: Sponsor Attachment; • 2442_Kambhampati-ONR-Final-Submitted-special-ai.pdf, Category: Sponsor Attachment; • 2840_ONR-Kambhampati-cdm-Final.pdf, Category: Sponsor Attachment; • In Person consent form, Category: Consent Form; • IRB protocol, Category: IRB Protocol; • Kambhampati-citiCompletionReport3499480.pdf, Category: Non-ASU human subjects training (if taken

Page 1 of 2

Figure 7.1: IRB approval page-1

	within last 3 years to grandfather in); • mudit_citi.pdf, Category: Non-ASU human subjects training (if taken within last 3 years to grandfather in); • Online Consent Form, Category: Consent Form; • participant_instructions.pdf, Category: Participant materials (specific directions for them); • Questionnaire.pdf, Category: Measures (Survey questions/Interview questions /interview guides/focus group questions); • recruit_online.pdf, Category: Recruitment Materials;
--	--

The IRB approved the protocol from 8/28/2023 to 8/27/2025 inclusive. Three weeks before 8/27/2025 you are to submit a completed Continuing Review application and required attachments to request continuing approval or closure.

If continuing review approval is not granted before the expiration date of 8/27/2025 approval of this protocol expires on that date. When consent is appropriate, you must use final, watermarked versions available under the “Documents” tab in ERA-IRB.

In conducting this protocol you are required to follow the requirements listed in the INVESTIGATOR MANUAL (HRP-103).

Sincerely,

IRB Administrator

cc: Mudit Verma
 Mudit Verma
 Subbarao Kambhampati

Figure 7.2: IRB approval page-2