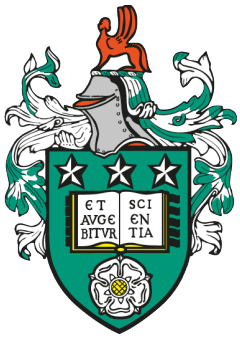
Robot Adaptability to People with Different Capabilities through Reinforcement Learning



Chengke Sun

The University of Leeds
School of Computing

Submitted in accordance with the requirements for the degree of
Doctor of Philosophy

May 2024

Declaration

The candidate confirms that the work submitted is his own, except where work which has formed part of jointly authored publications has been included. The contribution of the candidate and the other authors to this work has been explicitly indicated below. The candidate confirms that appropriate credit has been given within the thesis where reference has been made to the work of others.

Some parts of the work presented in this thesis have been published in the following article:

Sun, C., Cohn, A., & Leonetti, M. (2023). Online Human Capability Estimation through Reinforcement Learning and Interaction. In Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)

The above publication is primarily the work of the candidate.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my advisors, Anthony G Cohn and Matteo Leonetti. Their professional guidance, unwavering support, and deep friendship have been invaluable. Although the rare COVID epidemic posed challenges for us, my PhD journey is still exciting and meaningful under their support.

To my examiners, Mehmet Dogar and Matthew Howard. I deeply appreciate the thoughtful feedback received during my defense. It is an honor to engage in such intellectual discourse with you. Your feedback and comments not only helped me enrich the quality of my thesis but also gave me courage and confidence in my academic journey.

My time at the Robotics Laboratory of the University of Leeds has been truly incredible. I would like to thank the roboticists I worked with: Rafael Papallas, Wisdom Agboh, Alexia Toumpa, Ricardo Luna, Lipeng Chen, Hasan Mohammed, Logan Dunbar, Shengyin Wang, Zisong Xu, and Shaokang Wu. Although we don't have much time to work together due to the pandemic, I am very grateful for their friendship, support, discussions, and sharing of world cuisines.

Finally, I would like to thank my family. To my loving parents, Xuefeng and Jiang: thank you for your solid support regarding both my studies and life, and for your generous financial support. To my beloved wife, Hang, thank you for always being there for me no matter what.

Abstract

Robots are increasingly envisioned to offer sensible assistance across diverse settings. This thesis primarily centers on human-robot collaboration involving individuals with potential physical limitations. We aim to enable robots to perceive and adapt to human capabilities acutely, thereby offering personalized assistance. We present a framework for the online estimation of human capabilities based on Reinforcement Learning and Bayesian inference, including a series of estimation strategies and capability-guided exploration algorithms.

During Reinforcement Learning, the agent accumulates evidence that permits the continuous updating of human capability estimates via Bayesian inference. These estimates are pivotal in optimizing robot behavior. By modeling human capabilities as preconditions for specific robot actions, the agent can either deactivate the action or refer to the most suitable actions derived from a pre-training policy aligned with capability estimates.

We present three human-robot collaboration experiments to demonstrate and validate our framework: two are operated within a simulation environment, and one is a real-world experiment. The results show that our methodology accurately estimated human capabilities in static and dynamic environments. Furthermore, the robot exhibited faster adaptation and enhanced performance when human capabilities were incorporated into the learning process.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Service robots are expected to participate in the general public's lives widely. People are no longer satisfied with robots providing information and services to humans in a one-way manner. Instead, there is a growing desire for more collaborative interactions between robots and humans. This paradigm shift has given rise to human-robot collaboration tasks. While these tasks are often observed in controlled environments, such as robot competitions and research laboratories, achieving seamless integration of service robots into the daily routines of individuals remains a challenge.

In human-robot collaboration tasks set in real-world environments, the frequent assumption of able-bodied collaborators, often made in laboratory or controlled settings, becomes no longer viable. Service robots must not only perceive the capabilities of their human partners, but also incorporate the understanding into their high-level task-planning processes. It allows them to adjust their behavior based on the human's capability and improve social acceptability. A typical environment of this challenge arises in care home settings, where residents may exhibit varying physical capacities due to chronic illnesses, injuries, or age. Therefore, this thesis aims to advance the development of user-adaptive robots. These robots would be adept at offering timely, personalized assistance to their human collaborators.

Reinforcement Learning (RL) introduces a paradigm for robots to enhance their behaviors through interactions with the environment. This dynamically changing behavior improves the robot's adaptability, particularly when dealing with unknown collaborators in real-world settings. In the context of value-based RL algorithms used for online user adaptation, there are typically two causes behind policy changes: 1. When the agent performs actions that are not suitable for the current user, it accumulates negative rewards over time. Because these undesirable actions yield lower value compared to other actions, the agent gradually shifts its policy towards

actions with higher expected value; 2. The agent discovers actions with higher expected value than those currently in use through explorations. However, frequently trying low-value actions can be annoying to the human collaborator or even lead to task failure. Moreover, over-dependence on random exploration will cause the robot to perform risky actions. Overall, policy improvement that only relies on the above two principles pays no attention to human capabilities.

We present a framework that combines Bayesian inference and Reinforcement Learning to online estimate human capabilities in this thesis. These human capabilities are latent variables not directly observable by the agent. By introducing capability estimation into the RL process, the robot achieves a more adequate understanding and adaptation to its human collaborator. This early understanding of the capabilities supports the agent in improving its policy, thereby enhancing the quality of collaborative outcomes. Importantly, our proposed framework is model-free, which not only simplifies its implementation but also avoids potential model bias issues. This approach also prevents the computational intensive models like Partially Observable Markov Decision Processes (POMDPs).

Numerous approaches have been deployed to building user-adaptive robots, including statistical inference, POMDP planning, and user demonstration. Nevertheless, our method introduces capability estimation as a side effect of the RL context. While it is important to acknowledge that Reinforcement Learning is not the only method for estimating capabilities, our focus lies on the robot's ability to autonomously acquire this knowledge through its own interactions, without specialized calibration or input from the collaborator.

1.1 Main themes

This thesis mainly focuses on online human capability estimation and producing user-adaptive behaviors, thereby unfolding the following themes:

1. **The Role of Human Capabilities in Human-Robot Collaborations.**

   We model human capabilities as preconditions of robot actions, especially such actions involving human-robot interaction (HRI). For instance, a museum scenario where a robot serves as a tour guide allowed to drive fast. For effective and user-friendly guiding, the robot must remain within the visual range of the human collaborator. Thus, the collaborator must walk fast enough to follow the robot. However, this person may need more time to walk due to some health conditions,

such as using crutches. In such scenarios, the human's capability to walk quickly becomes the necessary precondition for the robot's fast-driving action.

When the human partner does not meet this precondition, a sensible alternative for the robot is to adjust its speed downward. Traveling at different speeds based on human capabilities is a form of adaptive and personalized assistance the robot offers in this collaborative task. Suppose the robot cannot sense the difference in this "walking" capability, the robot may drive to an inappropriate position, such as being too close to the collaborator or to a place where the collaborator cannot see it, leading to user-unfriendly and annoying feedback from users or task failures. Therefore, when modeling collaborative tasks, human capabilities are essential prior knowledge.

2. **Tracking Multiple Capabilities in Collaborative Tasks.**

   We treat human capabilities as distinct labels, thus, the capability estimation is to learn a predictive model. This model takes a feature vector as its input and produces a set of capability labels as its output. Given that collaborative tasks frequently require predicting various human capabilities, we approach human capability estimation as a multi-label learning problem. The model's output is a series of capability beliefs, reflecting the classifier's confidence in each capability.

3. **Exploring Correlations Between Human Capabilities and Observable Variables in the Context of Reinforcement Learning.**

   Human capabilities, as human properties, are latent variables, which means these variables cannot be directly observed but can be estimated based on a series of observed variables. Before building the predictive model, we need to determine the correlation between human capabilities and other observable variables within the Reinforcement Learning context. This study is essential because correlations are crucial to understanding and modeling the interrelationships between random variables. It helps us better understand and improve the accuracy of predictions. We propose three models with different correlations in this work. These distinct correlations subsequently guide us to three unique estimation strategies.

4. **Online Capability Estimation in Reinforcement Learning.**

   We addressed the multi-label learning problem using the Label Powerset method. For our Reinforcement Learning, both table-based Q-learning and Deep Q-Network (DQN) were adopted. Both Q-learning and DQN operate as model-free algorithms, indicating that the agent refines its policy through interactions

with its environment. Once the capability estimation model is well-trained and operational for predictions, we deploy the model in online RL, in which the model can accumulate evidence. Bayesian inference is then used to continuously update the probabilities of capabilities with the chosen estimation strategy.

Our capability estimation is integrated as middleware into both the Q-learning and DQN algorithms, which only slightly modify the existing Reinforcement Learning pipeline. This implementation enables the rapid adaptation of the capability estimation to other Reinforcement Learning methods.

5. **Capability-guided Exploration in Reinforcement Learning.**

   As we define capabilities as preconditions for robot actions, having capability estimates allows the agent to dynamically activate or deactivate certain actions based on the probabilities of associated capabilities. For instance, in the earlier tour guide scenario, when the probability of the capability to "walk quickly" drops below a predetermined threshold, the robot can decide to disable the fast-driving action.

   Furthermore, we obtain a collection of optimal policies during the training phase of building the prediction model. We can offer a structured exploratory path for the Reinforcement Learning agent by aligning these pre-trained policies with capability estimates.

6. **Applicability in Static and Dynamic Domains.**

   Given the varied environment of human-robot collaborative tasks, our framework must exhibit robustness in both static and dynamic environments. We conducted three experiments to evaluate both the performance of capability estimation and the Reinforcement Learning performance in various environments.

   In the first experiment, we tasked the robot with guiding a collaborator from a fixed starting location to a fixed goal, a collaborative navigation task involving two human capabilities. The second experiment centered on a manipulation task, where the robot and a human partner sort objects on a table. This task also considered two human capabilities. Nevertheless, this environment introduced uncertainty as the layout of objects on the table was randomly generated at the beginning of the task. Consequently, the state space became more complex than the first navigation task. Our final experiment occurred in a real-world environment, where the robot collaborated with a human partner to locate target objects dispersed across multiple rooms. Similar to the second task, the placement

of target objects was randomized. This task involved four human capabilities. The results of these experiments conclusively demonstrated that our framework accurately and robustly estimates capability probabilities.

1.2 Contributions

- **Paradigms for Correlating Human Capabilities with Observable Variables in Standard MDPs.**

  We present three estimation strategies, EST-Action, EST-State-Policy, and EST-Policy, to capture correlations between human capabilities and random variables in standard Markov Decision Processes (MDPs). These strategies are adaptable to collaborative tasks of varying complexity while remaining model-free.

- **Online Estimation of Human Capabilities in Model-Free Reinforcement Learning.**

  We introduce a framework for the online estimation of human capabilities within model-free Reinforcement Learning algorithms. This framework unfolds in three stages: pre-training, optional sampling, and deployment, seamlessly integrating with the RL pipeline with minimal modifications. We tested the Q-learning and DQN implementation. However, our approach can be adapted to other RL algorithms. We used chunk-based compression techniques to reduce the demands of storage and memory during pre-training and sampling.

- **Capability-Guided Exploration in Reinforcement Learning.**

  We present two capability-guided exploration strategies, RLC and RLC-Policy, as alternatives to pure random exploration in Reinforcement Learning. Human capabilities are defined as preconditions for robot actions. RLC uses this prior knowledge and the results of capability estimation, while RLC-Policy further incorporates the pre-trained policy derived during the training state of capability estimation.

- **Implementation of Human-Robot Collaboration Experiments.**

  We present three collaborative tasks for the need to evaluate the human-capability adaptive learning framework. Two of these tasks are simulated, involving robot navigation and manipulation. The navigation task represents static domains, requiring the robot to guide a collaborator from a fixed starting point to a fixed

endpoint. In the manipulation task, uncertainty and dynamic environments are introduced, with the robot and collaborator jointly sorting randomly placed objects on a table. The third task operates in the real world and is a variation of the Scavenger Hunt Game benchmark. This task challenges the robot and collaborator to locate multiple randomly positioned objects within various rooms.

1.3 Thesis Outlines

Chapter 2 introduces the background of this thesis. Chapter 3 introduces related work. Chapter 4 introduces the estimation strategies and the estimation framework. Chapter 5 introduces the experimental setup and evaluations. Chapter 6 summarizes the experimental results, limitations, and introduces the future work.

Chapter 2

Background

2.1 Markov Decision Process

Task planning for robots can be viewed as a sequential decision-making problem in an uncertain system. Markov Decision Process (MDP) is a model that describes sequential decision-making and rewards in a system where the Markov property is met. The Equation (2.1) illustrates the case where Markov property is satisfied, that is, the transition probability of current state s_{t+1} depends only on the previous state s_t rather than the state history:

$$P(s_{t+1} \mid s_0, s_1, \ldots, s_t) = P(s_{t+1} \mid s_t). \tag{2.1}$$

A MDP is represented as a tuple $D = \langle S, A, P_a, R_a \rangle$ to describe the stochastic process, where S is a set of states, A is a set of actions, $P_a(s, s')$ is the probability of state transition from the state $s = s_t$ to the next state $s' = s_{t+1}$ by executing the action a:

$$P_a(s, s') = P(s_{t+1} = s' \mid s_t = s, a_t = a). \tag{2.2}$$

$R_a(s, s')$ is the reward function, representing the immediate reward the agent received after transitioning from state s to the next state s', due to action a.

MDP is the foundation of Reinforcement Learning that we will introduce next. The random variables present in MDP shape the feature space for our capability estimation.

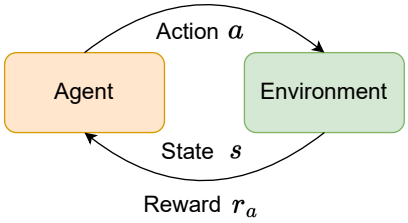


Fig. 2.1 The diagram of Reinforcement Learning

2.2 Reinforcement Learning

Reinforcement Learning (RL) is a machine learning method that enables an agent to learn optimal behaviors through observations and interactions between the agent and the environment. MDP is a formal representation of such an environment. As diagrammed in Figure 2.1, the agent is in an interactive environment described by the state space S, and the environment is observable for the agent by perceiving the state $s \in S$. The agent can choose an action $a \in A$, and then the environment rewards the agent based on the reward function R_a. The goal of the RL agent is to learn a policy π, which indicates the agent to perform an action a given a state s: $a = \pi(s)$. The Equation (2.3) shows the deterministic policy. The Equation (2.4) shows the stochastic policy, where $\pi(a \mid s)$ represents the probability to choose the action a given the state s:

$$\pi : A \times S \to [0, 1] \tag{2.3}$$

$$\pi : A \times S \to \mathbb{R}. \tag{2.4}$$

The goal of the agent is to maximize the expected cumulative reward, represented by the return $\mathcal{R}(\tau)$, where the trajectory τ denotes a sequence of states and actions. A policy is considered optimal if its return is greater than or equal to the return of any other policy, for all states.

One kind of return is the n-steps undiscounted return:

$$\mathcal{R}(\tau) = \sum_{t=0}^{T} r_t. \tag{2.5}$$

Another kind of return is the n-steps discounted return, where γ is the discount factor:

$$\mathcal{R}(\tau) = \sum_{t=0}^{T} \gamma^t r_{t+1}. \tag{2.6}$$

In the rest of this thesis, the expected return is considered as the n-steps discounted return. We introduce the state value function $V^\pi(s)$ that denotes the expected return if the agent starts in state s and follows the policy π:

$$V^\pi(s) = \mathbb{E}_{\tau\sim\pi}\left[\mathcal{R}(\tau) \mid s_0 = s\right]. \tag{2.7}$$

The state-action value function $Q^\pi(s,a)$ denotes the expected return if the agent starts in state s, takes an action a, and then follows the policy π:

$$Q^\pi(s,a) = \mathbb{E}_{\tau\sim\pi}\left[\mathcal{R}(\tau) \mid s_0 = s, a_0 = a\right]. \tag{2.8}$$

An optimal policy π^* maximizes the expected return:

$$\pi^* = \arg\max_\pi \sum_{s\in S} V^\pi(s). \tag{2.9}$$

Similarly, we can define the optimal value function $V^*(s)$ and $Q^*(s,a)$ when the optimal policy π^* is applied:

$$V^*(s) = \max_\pi \mathbb{E}_{\tau\sim\pi}\left[\mathcal{R}(\tau) \mid s_0 = s\right] \tag{2.10}$$

$$Q^*(s,a) = \max_\pi \mathbb{E}_{\tau\sim\pi}\left[\mathcal{R}(\tau) \mid s_0 = s, a_0 = a\right]. \tag{2.11}$$

2.2.1 Model-based Learning

In model-based learning, the Markov Decision Process (MDP) is either known to the agent or the agent actively learns a model of the environment. That is, the agent knows the probability of state transition P_a and the reward function R_a as well. Based on this assumption and the Markov property, the expected maximized values in Equation (2.10) and (2.11) can be further solved when both state space and action space are finite:

$$V^*_\gamma(s) = \max_{a\in A} \sum_{s'\in S} P^a_{s\to s'}(R^a_{s\to s'} + \gamma V^*_\gamma(s')) \tag{2.12}$$

$$Q^*_\gamma(s) = \max_{a\in A} \sum_{s'\in S} P^a_{s\to s'}(R^a_{s\to s'} + \gamma \max_{a'\in A} Q^*_\gamma(s',a')). \tag{2.13}$$

The above equations (2.12) and (2.13) are also called Bellman Optimality Equations, which are the foundation of typical model-based RL methods such as policy iteration

and value iteration. While this method is relevant, it will not be further used in this work because our work is model-free, which will be explained in the subsequent section.

2.2.2 Model-free Learning

The state transition P_a and reward functions R_a in MDP are usually hard to know in actual tasks. The model-free learning algorithms allow the agent to learn the policy without being aware of state transition and reward functions. Model-free learning methods can be divided into two categories: Value-based Learning and Policy-Based Learning. The Value-based approaches learn the state-action value function (also named value function or Q-function) $Q^*(s, a)$. The Policy-Based methods directly learn the policy function $\pi(a \mid s)$.

The Monte Carlo (MC) method is a model-free learning algorithm that replaces explicit policy evaluation by learning the approximate value function based on multiple sampling of the complete trajectory.

Q-learning is a Temporal-Difference (TD) method. Compared with MC, the TD method updates the value function estimation at every step, which is more efficient. The optimal Q-function obeys the following Bellman Optimality Equation:

$$Q^*(s, a) = \mathbb{E}_{s' \sim P_a} \left[R_a(s, a) + \gamma \max_{a'} Q^*(s', a') \right]. \tag{2.14}$$

In scenarios where both the state space and action space are finite, the Q-function can be conveniently represented using a table structure (shown in Figure 2.2). The size of the table is determined by $|S| \times |A|$. Each entity (Q-value) in the table corresponds to a specific state-action pair and represents the estimated value of that action in the given state. At the beginning of the training, a table $\tilde{Q}$ is initialized with any values. Then, by updating $\tilde{Q}$ during the learning process, the $\tilde{Q}$ converges to Q^*. The Algorithm 1 outlines the iterative process of updating the Q-values based on observed rewards and estimated future rewards.

Q-table			
	a_0	a_1	a_2
s_0	$Q(s_0, a_0)$	$Q(s_0, a_1)$	$Q(s_0, a_2)$
s_1	$Q(s_1, a_0)$	$Q(s_1, a_1)$	$Q(s_1, a_2)$

Fig. 2.2 An example of Q-table where $S = \{s_0, s_1\}, A = \{a_0, a_1, a_2\}$

In Reinforcement Learning, Behavior Policy is an agent's policy to select actions in a given state. The primary role of the behavior policy is to explore the environment

Algorithm 1: Q-Learning

Input: the environment E, the number of learning episode *episodes*, the learning rate α, the discount factor γ

Output: value function $Q(\cdot)$

$Q(s,a) = 0$ for all $s \in S$ and $a \in A(s)$
for $i \leftarrow 1$ ***to*** *episodes* **do**
  Observe initial s_0 from E
  $s \leftarrow s_0$
  repeat
    Choose action a_t using policy derived from Q (e.g., ϵ-greedy)
    Take action a and observe r, s' from E
    $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$
    $s \leftarrow s'$
  until s_t *is terminal*;
return Q

and gather experience by interacting with it. Target Policy is the policy the agent aims to learn or optimize based on the experience it receives.

On-policy learning algorithms evaluate and improve the same policy used to select actions. In other words, these algorithms employ the same policy in both Behavior Policy and Target Policy. In contrast, Off-Policy learning algorithms employ different policies for evaluating and action selection. Q-learning is an Off-Policy method since it uses ϵ-greedy for action selection whereas greedy policy is for evaluating. The ϵ-greedy shows in (2.15), in which the agent chooses optimal action given the state s with probability $1 - \epsilon$ and chooses a random action from A with probability ϵ. The Off-policy methods balance the two behaviors of exploring the reward of each action and exploiting the action with the highest current reward, thus preventing the agent from becoming trapped in a local optimum:

$$\pi(s) = \begin{cases} \arg\max_a Q(s,a), & P_r = 1 - \epsilon \\ \texttt{rand}(A), & P_r = \epsilon. \end{cases} \tag{2.15}$$

The Softmax method is another algorithm for selecting actions, transforming a set of Q-values into a probability distribution. Instead of only performing action with the highest reward, Softmax assigns a non-zero probability to all actions, even those with lower estimated values, and the probability of each action is proportional to its estimated value. The algorithm of Softmax is shown in Equation (2.16), where t denotes the positive parameter called temperature. For high t, probabilities of all

actions being chosen converge gradually, and the specificity of actions will decrease. For low t, the probability of the action with the highest reward being chosen tends to be 1:

$$P(a \mid s) = \frac{\exp \frac{Q(s,a)}{t}}{\sum_{a_i \in A} \exp \frac{Q(s,a_i)}{t}}. \tag{2.16}$$

The Reinforcement Learning algorithms involved in our work are Off-policy and model-free. Q-learning is one of our implementations in capability estimation. The Softmax method is applied to transform action values into probabilities.

2.3 Deep Reinforcement Learning

In the previous Section 2.2, we introduced Reinforcement Learning methods that rely on finite state and action spaces. Such approaches focus on identifying the optimal action for each discrete state and typically store policies in tables or databases. However, these methods have limited generalization abilities and pose challenges when dealing with large or continuous state spaces, requiring extensive storage capacity.

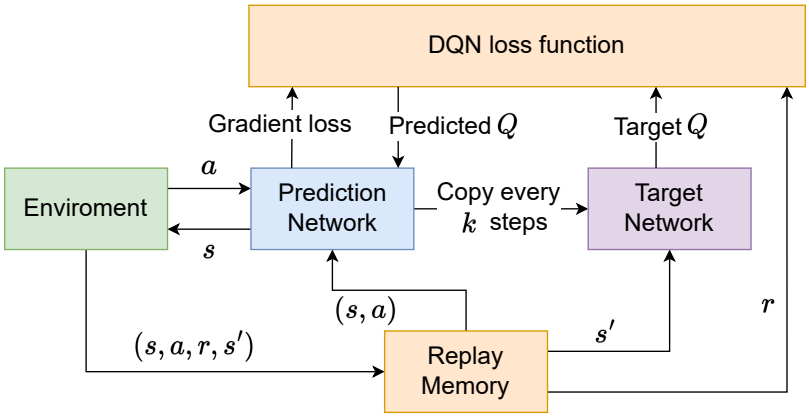


Fig. 2.3 A diagram of the DQN algorithm

Deep Reinforcement Learning methods, such as the Deep Q-Network (DQN)[39], use deep neural networks (DNN) to approximate the value function Q^*, denoted as $Q(s, a, \omega)$, where ω represents the learned weights of DNN during training. Figure 2.3 shows the data flow of DQN. The primary objective is to train $Q(s, a, \omega)$ to closely approximate Q^*. The output of DQN is a $|A|$-dimensional vector. Each element in this vector corresponds to the estimated value of a specific action. The generalization

ability of DQN arises from the ability of DNN to approximate complex functions and capture high-dimensional patterns in the state space. The neural networks receive state representations from the agent's environment, and transform them through multiple layers of interconnected neurons. These layers progressively extract and abstract relevant features and patterns, capturing the underlying structure and dynamics of the environment. As DQN learns from a diverse set of experiences and updates ω accordingly, it gradually builds a rich internal representation of the environment. Such a representation serves as a powerful tool for generalization, allowing DQN-based agents to transfer their acquired knowledge and competencies to new, unseen environments or tasks, enhancing their problem-solving abilities beyond the specific contexts they were trained in.

Similar to Q-learning, one limitation of DQN is overestimating Q-values[22][23], which can lead to suboptimal performance. In Double DQN presented by Hasselt et al.[23], the target network is still used to select the action with the maximum Q-value, but the main network is employed to estimate the Q-value associated with that action. By separating the action selection and Q-value estimation, Double DQN mitigates the overestimation problem.

We used DQN as one of the implementations for our capability estimation framework in this thesis.

2.4 Multi-Label Classification

Classification tasks can be mainly divided into three paradigms: binary classification, multi-class classification, and multi-label classification, also known as binary learning, multi-class learning, and multi-label learning. The objective of binary classification is to train a model to determine whether an instance belongs to a specific class. Multi-class classification expands the binary classification approach by generating more than two distinct classes or categories as the output. Binary classification and multi-class classification are categorized as single-label learning problems, where each instance is associated with a single label. This distinction sets single-label learning apart from multi-label learning, where objects can be associated with multiple labels simultaneously.

Solving multi-label learning is more challenging than single-label learning because of the computational and query costs associated with the exponentially growing number of label combinations and high-dimensional feature vectors. However, the

critical advantage of multi-label classification is that it considers the dependencies and correlations between labels.

Methods to address the multi-label learning problem can be broadly classified into three categories: First-order strategy, Second-Order strategy, and High-Order strategy [64][65]. The first-order strategy breaks down the multi-label learning problem into separate binary learning tasks, ignoring the potential correlations between labels. In contrast, the second-order strategy considers the relationships between pairs of labels. However, this approach can sometimes be hindered by the specific combination or sequencing of label pairs. The high-order strategy, on the other hand, recognizes correlations across all labels, granting superior correlation-modeling and generalization abilities compared to the first and second-order strategies. However, this strategy is computationally intensive and less scalable.

Label Powerset (LP) transformation [59] is a representative example of a High-Order strategy. This method tackles the multi-label learning problem by creating new classes for each unique combination of labels, thereby transforming it into a multi-class learning problem. LP method is simple and effective but is vulnerable to an imbalanced dataset[18]. Furthermore, the number of label combinations will be up to 2^n for n labels, making it more suitable for scenarios with a smaller number of labels.

The above three strategies are categorized as problem transformation methods. They work by transforming the multi-label learning problem into a form of the single-label learning paradigm. In contrast, another category of approaches to address the multi-label learning problem is the algorithm adaptation methods, which directly tweak existing single-label learning algorithms to address the multi-label learning problem.

Estimating human capabilities in human-robot collaborative tasks can be viewed as a multi-label learning problem. Because in practical collaborative scenarios, the robot frequently has to understand and adapt to various dimensions of human capabilities. We use the Label Powerset method in this work to address the multi-label learning problem.

2.5 Naive Bayes Classifier

In Section 2.4, we discussed the application of multi-label learning for estimating human capabilities. This process is subsequently transformed into single-label learning using the LP method. However, to effectively tackle this single-label learning challenge, the selection of appropriate methods or techniques is crucial.

A widely adopted single-label learning algorithm is the Naive Bayes (NB) classifier. The NB classifier is frequently applied in various domains due to its simplicity and efficiency. We assume that L is the set of label values and $\mathbf{x}$ is the feature vector. According to Bayes' theorem, the posterior probability $P(l \mid \mathbf{x})$ that the feature vector $\mathbf{x}$ belongs to class l is shown in Equation (2.17), where $P(l)$ is the prior probability, $P(\mathbf{x} \mid l)$ is the likelihood, and $P(\mathbf{x})$ is the marginal probability:

$$P(l \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid l)P(l)}{P(\mathbf{x})}. \tag{2.17}$$

The NB method assumes all features in $\mathbf{x}$ are conditionally independent, resulting in the likelihood can be expressed as Equation (2.18). By applying the total probability rule, Equation (2.17) is updated to Equation (2.19):

$$P(\mathbf{x} \mid l) = P(x_1 \mid l)P(x_2 \mid l) \cdots P(x_n \mid l), \tag{2.18}$$

$$P(l \mid \mathbf{x}) = \frac{\prod_{x \in \mathbf{x}} P(x \mid l)P(l)}{\sum_{l' \in L} \prod_{x \in \mathbf{x}} P(x \mid l')P(l')}. \tag{2.19}$$

Since the denominator of Equation (2.19) is equal for all classes, the class to which the feature vector $\mathbf{x}$ most likely belongs is shown in Equation (2.20):

$$h_{nb}(\mathbf{x}) = \arg\max_{l \in L} P(l) \prod_{x \in \mathbf{x}} P(x \mid l). \tag{2.20}$$

However, suppose events within the feature vector were not observed during training. In that case, these events will be assigned a zero probability, causing the whole Equation (2.18) to be 0, and the classifier cannot continue processing. From another perspective, events that do not appear in the training set are not entirely impossible. Smoothing algorithms are applied to solve the zero probability problem and improve computational accuracy. The Equation (2.21) shows Laplace smoothing, where $count(x)$ is the number of occurrences of event x in the training data, α is the smoothing factor (generally assigned with 1), N is the total number of observations, and $|L|$ represents the number of classes:

$$P_{LAP}(x) = \frac{count(x) + \alpha}{N + |L| \times \alpha}. \tag{2.21}$$

When calculating the product of multiple tiny probabilities, it is possible for the result to approach zero, which can lead to numerical underflow issues. A common approach to address this issue is to migrate the probabilities to the logarithmic space,

as shown in Equation (2.22). By transforming the tiny probabilities to logarithmic values, the multiplication operation is replaced by addition, which helps prevent the occurrence of extremely small values that can cause underflow:

$$P(\mathbf{x} \mid c)P(l) = \sum_{x \in \mathbf{x}} \log P(x \mid l) + \log P(l). \tag{2.22}$$

2.6 Bayesian Network

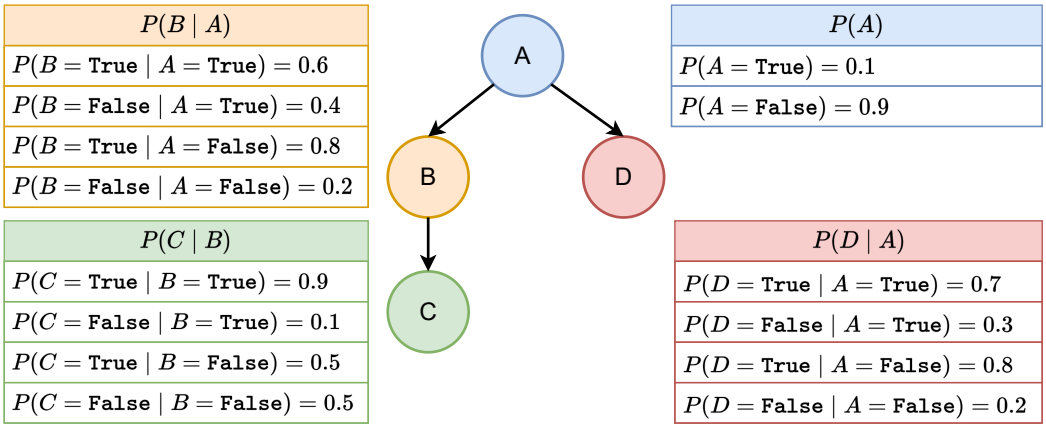


Fig. 2.4 An example of Bayesian network

A Bayesian network is a probabilistic graphical model used to represent and reason about uncertain knowledge. It consists of a directed acyclic graph (DAG) where nodes represent random variables, and edges represent conditional dependencies between them. Each node in the network is associated with a probability distribution that reflects the uncertainty in the variable's value given its parents in the graph. The quantitative conditional dependencies are represented by the Conditional Probability Table (CPT). Figure 2.4 shows an example of a Bayesian network. We define a Bayesian network as a tuple $\langle \mathbb{G}, \Theta \rangle$, where $\mathbb{G}$ denotes the DAG structure and Θ denotes CPTs between associated random variables. The MDP model (ct. Section 2.1) can be represented as a Bayesian network, as shown in Figure 2.5.

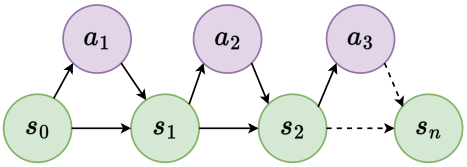


Fig. 2.5 The Bayesian network of MDP

The Bayesian network effectively represents the conditional independence between random variables. By constructing a Bayesian network, we can capture the interactions between various factors that contribute to estimations of human capabilities, and assess their conditional dependencies. As shown in Figure 2.4, we can deduce that the random variables B and D are conditionally independent when the random variable A is given, denoted as $B \perp\!\!\!\perp D \mid A$. Knowing the value of A eliminates the dependence between B and D, and their relationship can be explained solely by their common dependence on A. In other words, once we have the value of A, the knowledge of B does not provide any additional information about D, and vice versa. We can further derive that the joint probability of the random variables in Figure 2.4 is $P(A, B, C, D) = P(A)P(B|A)P(D|A)P(C|B)$.

The Bayesian network is a fundamental tool in this work, helping us understand the correlations among random variables.

Chapter 3

Related Work

3.1 User adaptability

In this section, we investigate the definition of user adaptability, as the foundation of our research. User adaptability of service robots has been widely researched in the robotic community because users prefer robots that respect human conditions and preferences [47]. User-adaptive robots aim to assist people by adapting their behavior to meet users' intentions, preferences, and personalities [24]. Furthermore, user adaption also plays a significant role in building socially acceptable robots [53].

Adaptive systems are commonly characterized as either localized or personalized [24]. Localization pertains to adaptive systems considering broad aspects, such as regional or cultural characteristics. Conversely, personalization involves the narrow scope of adapting, like the capabilities and preferences of a group of users or a specific individual. Preferences have been organized in a taxonomy for physically assistive robots involving human capabilities as parameters that affect how robots' actions are performed [10].

While human capabilities and preferences are frequently examined collectively, we believe that they merit separate consideration instead of merely treating capabilities as parameters of preferences. Human capabilities are objective, involving the physical and cognitive skills an individual possesses, from fundamental motor abilities such as walking and object manipulation to more cognitive expertise such as problem-solving, language interpretation and expression, emotional understanding, and the ability to learn. Human preferences, on the other hand, are subjective and individual-specific. They often refer to the decisions made given a variety of options. This decision-making process could be influenced by personal experiences, social surroundings, cultural backgrounds, inherent personality traits, and emotions. Common topics involving

collaborators' preferences include food choices, favored activities, and the order of arranging objects.

Finally, let us examine the definition of adaptability through the output of adaptive systems. Martins et al. define adaptability as the ability of a system to automatically modify its operating parameters accordingly to perform functions [36]. Mehdi et al. conceptualized user adaptability as the robot's capacity to adjust its parameters in response to collaborator information [24]. Thus, the target of user adaptation can be specific parameters of end effectors, such as the mobile robot's movement speed or the speaker's volume. More importantly, user adaptation also applies to more abstract layers in the robotic systems, such as personalized behaviors and high-level task planning among multiple agents. This thesis mainly focuses on high-level single-agent user adaptation with personalization.

3.2 System Structure

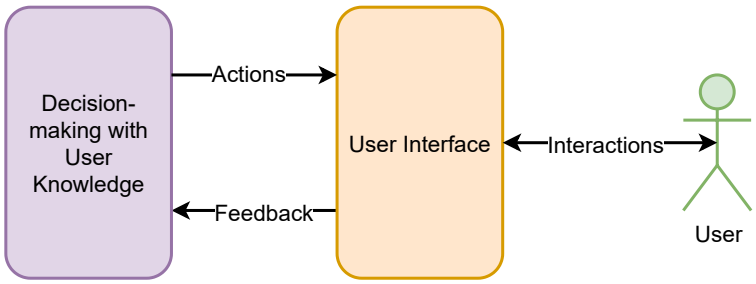


Fig. 3.1 General structure of user-adaptive robotic system

For effective user adaptation in robotic systems, one must take into account the roles played by the robot, the user, and the environment. Figure 3.1 shows the general structure of the user-adaptive robotic system. In human-robot collaborative tasks, the robot typically serves as the user interface, equipped with sensors to observe information and actuators to perform actions. The decision-making component, commonly embodied by a task or motion planner, processes the feedback and information received from the user interface to formulate adaptive actions or motion trajectories given the user knowledge. User-adaptive systems require knowledge of the user, interaction system, and task [42]. Therefore, user models are commonly integrated with the decision-making component. These user models provide insights into the unique characteristics of the user, enabling adaptive interactions between the robot and its collaborators. In the next section, we will delve into the taxonomy of user models.

3.3 User Model

User models serve as an explicit knowledge base, enabling adaptive systems to access the necessary information for localization and personalization. These models encode relevant user attributes, such as preferences and capabilities, to support the system to generate personalized behavior that matches the user's intentions and preferences, consequently boosting user satisfaction and acceptance. User models can take on various representations, ranging from pre-defined rules to probabilistic models that integrate representation with reasoning. It is interesting to track mutable user information and characteristics in human-robot interaction (HRI). Thus, inspired by studies [36] and [24], we categorize the persistence of user models into three types: Static User Model, Dynamic User Model, and Implicit User Model.

3.3.1 Static User Model

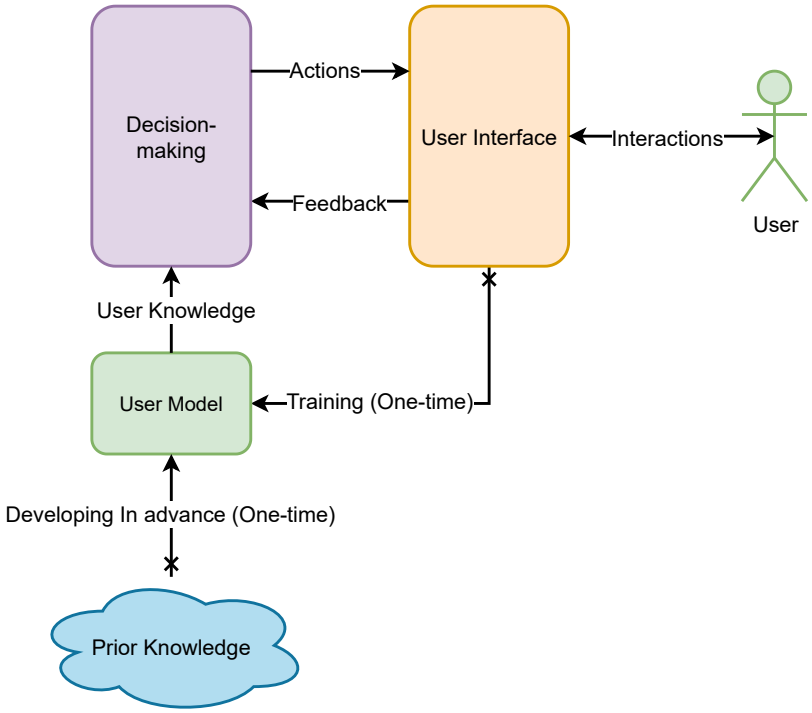


Fig. 3.2 The structure of static user model

Figure 3.2 shows the structure of the static user model. Static models are intuitive in user-adaptive systems, where information about the user is collected during or even before the interaction and remains fixed throughout, resulting in reasonable and

repeatable adaptions. This information can be obtained through feature extractions, the prior knowledge, and questionnaires.

Some service robots enhance user engagement and acceptance by observing user states and implementing pre-defined adaptive behaviors[21][38]. The decision process can be rule-based matching or inferred by finite state machines. Memorizing user characteristics and adapting conversation to the user's preferences in HRI can help improve user satisfaction. Churamani et al.[11] developed an adaptive robot that can generate personalized dialogues by training faces and voice recognition models, as well as memorizing personal information in the knowledge base.

Static user models can provide users with personalized assistance in human-robot collaborative manipulation. Klee et al.[31] present an assisting robot for dressing a user. The knowledge base of their system stores user constraints used to select motions adaptively. Gao et al. [17] present an approach to building kinematics models of the upper body. The robot can consider the user's moveability and produce adaptive motions to help the partner dress. In [48], the user model contains children's personal information and the history of dance moves, which supports producing personalized dance performances for kids.

Questionnaires can be used to train user models to learn user preferences. In [13], each participant needs to fill out a questionnaire about preferences, and the system will match users to predefined user profiles. The parameters of the robot's actions are adjusted according to the user's personas. In [3], the authors compared two techniques that establish users' personalities initially: questionnaires and linguistic models. A symbol-based reasoning system builds a user model based on the user's answers [8][9].

Robust prior knowledge is a solid foundation for causal analysis in user models and behavior optimization in robotic systems. In most collaborative tasks, the observation space and planning space are constrained by the environment, the scope of interactions, sensors, and actuators. For instance, if a collaborator struggles to read text on a display. An assistant robot, if solely programmed to read out text, would perform that action rather than modify the font size for better visibility. The robot does not consider issues such as reading disabilities or language barriers the collaborator faces, as they are irrelevant to the task. Hence, when the system observes that the collaborator struggles to read or when the robot takes on the task of reading aloud, it might reasonably infer a negative assessment of the collaborator's visual capabilities. Simultaneously, the predefined fallback action of reading the text can provide positive assistance to the collaborators. Our work applies expert knowledge to define preconditions for certain robot actions relevant to human capabilities.

3.3.2 Dynamic User Model

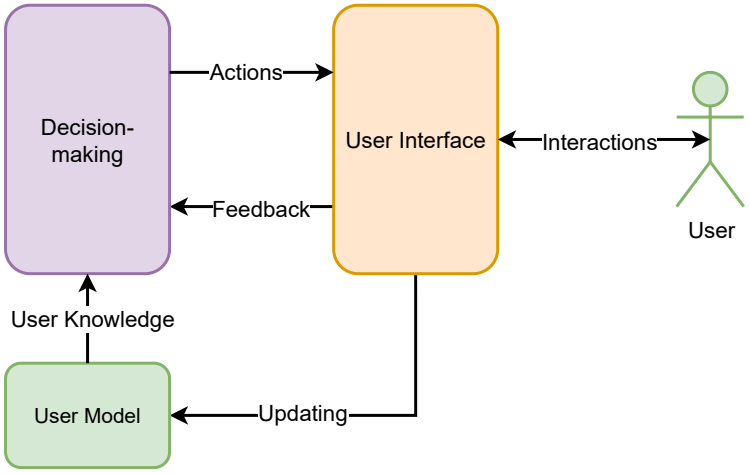


Fig. 3.3 The structure of dynamic user model

Figure 3.3 shows the structure of the dynamic user model. Dynamic user models facilitate the updating of knowledge pertaining to one or multiple users across successive interactions. The methods behind such models typically involve learning from interaction histories and tracking over extended periods. By continuously updating the user model, the system can adapt to shifting user characteristics and preferences throughout the interaction process.

A long-term adaptive dialog system implemented by representing user interaction history through probabilistic models [40]. Aylett et al [5]. present an empathic robot tutor that continuously tracks features that reflect the user's knowledge level to adjust relevant interactions. By modeling the limited historical steps of collaborators, the robot can provide flexible policies when humans show adaptation; otherwise, output policies are adapted to humans to gain trust [41]. In [12], the authors present a system that estimates the collaborator's mental states over interactions; meanwhile, the collaborators can only get partial information. Their robot provides the collaborator with the information he misses while minimizing the amount of communication to reduce interruptions. In [49], the authors introduce Stochastic Context-Free Grammars with synchronization-based adaptability. When a user indicates in an interaction that they do not like the music the agent selects, the user's preferences will be updated. Service robots working in shopping malls can provide users with personalized guidance services [27]. Their robot system stores personal information, conversation history, and guidance history in databases. Similarly, the robot reserves user profiles via a database,

such as capabilities, special needs, and preferences. Moreover, the robot automatically updates their preferences as seniors age [44].

Sekmen et al.[51] implemented a Bayesian learning system that employed questionnaires to build initial probabilities of the network. The questionnaire collected information such as the courses and meetings users participated in and their beverage preferences at different times of the day. In their adaptive system, the robot recommends beverages and reminds users to attend events in time based on their preferences. The user preferences model is updated after every interaction. In contrast, the non-adaptive version requires users to explicitly instruct the robot to set reminders. Based on user ratings obtained from experiments, the results demonstrated a preference for the adaptive system over the non-adaptive one.

In addition to tracking the user's interaction history, adaptability can be generalized to new users using user profiles and online analytics. Karami et al [28]. present an MDP-based learning framework that contains dynamic user profiles. These profiles can be initialized with domain knowledge and continue to be updated during interactions.

In contrast to the static model, which relies on fixed knowledge, the dynamic model's ongoing update mechanism in long-term collaborative tasks ensures more responsive user adaptability. Moreover, adaptive systems can directly optimize the robot's plan or policy by considering user characteristics as latent variables that influence decision-making, as an implicit user model. This approach eliminates the need for a large number of explicit user model components in the system structure.

3.3.3 Implicit User Model

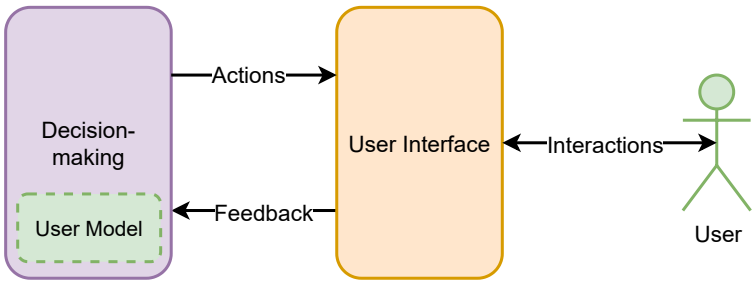


Fig. 3.4 The structure of implicit user model

Figure 3.4 shows the structure of the implicit user model. These systems can interact with users by using immediate feedback obtained during the interaction without explicitly storing knowledge about user preferences. Given the blurred boundary

between the user model and decision-making in such systems, we discuss them further in the next section.

3.4 Decision-Making with User Knowledge

Alongside considering the persistence and mutability of the user model, recent work focuses on the methodologies used for decision-making strategies combining user models. Beyond traditional symbolic reasoning, a plethora of machine learning techniques are now available for enhancing decision-making processes.

3.4.1 Rule-based Methods

Rule-based systems typically involve human-written rules, logic programming, search trees, and state machines. Such as finite-state automaton in [38] and pre-coded transitions of behaviors in [27]. The developer directly controls the adaptive behavior of these systems, exhibiting efficient adaptive decision-making in a controlled environment. However, hand-crafted rules often rely on interpretable natural language or database entries, limiting the ability to generalize and pose challenges when handling complex tasks. The system's adaptability to the environment is highly dependent on the robustness of expert knowledge and the low complexity of the task.

3.4.2 Supervised Methods

Supervised learning is a category of machine learning methods that involves training an algorithm on labeled data. Typical algorithms applied in user-adaptive robots include Naive Bayes, support vector machines (SVM), decision trees, and neural networks. Compared to rule-based methods, machine learning methods offer several advantages:

1. **Processing Complex Data:** Machine learning techniques excel at analyzing high-dimensional and continuous state space features. They efficiently process and interpret complex datasets, making them well-suited for adaptive challenges involving intricate data structures.

2. **Optimized Storage Efficiency:** Instead of relying on extensive, explicitly stored rules like rule-based systems, machine learning algorithms discern patterns from the data, resulting in condensed and effective representations. It is especially beneficial for robot systems with limited computational resources.

3. **Generalization Abilities:** Machine learning models are trained on existing data, allowing them to identify patterns and make informed decisions on new, unseen data. This generalization ability ensures that the models remain adaptive and effective in varied and evolving scenarios.

Several research studies have explored the application of SVM in different domains. For instance, in [3], the authors utilize SVM to predict a user's personality based on their utterances. Meanwhile, in [26], the authors implement SVM to predict which toppings customers will choose in a collaborative sandwich-making scenario, based on their gaze patterns. The features considered for prediction include the number of glances, duration of the first glance, total duration of gaze, and the most recently glanced item. SVM method can be computationally intensive and require careful tuning of hyperparameters for optimal performance.

Collaborative filtering is a popular machine learning technique employed in recommendation systems. It operates under the assumption that users who have agreed in the past will likely agree in the future about their preference for certain items. Thus, it predicts user's preferences according to patterns from users with similar tastes. This technique makes the pre-trained user model adapt to new users rapidly by matching known characteristics. Abdo et al. present a system that learns user preferences from pre-collected ratings and makes predictions using collaborative filtering [1]. However, this method has some drawbacks. One is the cold start problem, where new users or items have no or limited historical data available, making it difficult to provide accurate recommendations. Other challenges such as the sparsity problem, where the number of available ratings for items is much smaller than the total number of items, making it challenging to find similar users or entities for the recommendation.

Bayesian inference has been applied to estimate user intention [37]. Martins et al. introduce a user-adaptive service-selecting model using Bayesian programming [34], where the Bayesian inference can support agents to plan personalized behaviors by creating a probabilistic model that predicts collaborators' behavior based on their past behavior, environmental factors, and other relevant information. In this thesis, Bayesian inference does not yield the action executed by the robot, which is instead obtained through RL. It is used as a side-effect of RL, to estimate human capabilities as action preconditions. Martins et al. present a distributed learning system to infer the characteristics of users [35].

3.4.3 Reinforcement Learning Methods

Supervised learning is learning from a labeled training set. However, procuring a vast and representative set of learning samples in scenarios involving interactions with both the environment and humans is often challenging. Reinforcement Learning agents, in contrast, actively explore their environment, gaining experience and behavior norms. Their primary goal is to determine actions that maximize cumulative rewards. Consequently, Reinforcement Learning offers a flexible way to bridges the gap left by supervised learning, especially when labeled training data is either limited or arduous to acquire, which is a common challenge for robotic tasks.

The classic model-based RL learning methods are Policy Iteration and Value Iteration. Karami et al. [29] propose an adaptive system that models the user profile and activity in the MDP, where the robot analyzes every user's feedback to learn user preferences and update the reward function. Tseng et al. [56] propose an model-based RL with online reward shaping for HRI, where the reward shaping is used to enhance the reward from the human feedback.

Partially Observable Markov Decision Process (POMDP) is a mathematical framework that models decision-making problems where the system's state cannot be fully observed. POMDP extends the MDP by introducing the concept of observation, which captures the fact that the agent's knowledge of the state is incomplete. Taha et al.[54] propose a POMDP model encoding multiple HRI variables to infer user's intentions has been applied to an intelligent wheelchair. Broz et al.[7] introduced a time-indexed POMDP model that contains hidden user intentions, improving the quality of interactions. Their example domains are a simulated elevator-riding task and a simulated driving task. Michelangelo et al.[16] propose a work based on Mixed Observability Markov Decision Process (MOMDP), which uses hierarchical states to simplify the POMDP computation. Their system monitors user's movements and adjusts to their walking speed and engagement level. Lam et al.[32] presented a POMDP framework incorporating the human model, machine dynamics model and observation model to determine appropriate feedback from controllers, and overriding operations. They employed simulations to show the benefits of this framework. Wang et al. [62] formulated the decision-making process as a POMDP and presented a framework to infer the intention of human partners based on their movements. They verified their work on a human–robot table tennis domain with nonlinear and stochastic human behaviors. Görür et al. [20] presented an autonomous adaptive framework combined with the Anticipatory Partially Observable Markov Decision Process (A-POMDP) model for short-behaviors and a Bayesian Policy for long-term changing human characteristics,

where the experiment is a product inspection and storing job performed by humans and robots collaboratively in a simulated factory environment.

As the number of states increases, the number of possible belief states grows exponentially, making it difficult to compute an exact solution to the POMDP. Consequently, solving POMDP can be computationally intensive for problems with a large number of states, posing challenges in deriving optimal solutions within a feasible time frame. Our approach also infers latent variables, but does not require the POMDP modeling. Instead, we address the state space using a standard MDP, while estimating capabilities separately. Model-free learning is used online, avoiding complex POMDP planning.

In model-free Reinforcement Learning, the agent learns to make decisions based on trial-and-error interactions with the environment. Therefore, adaptive behavior can occur naturally through continuous interaction by receiving positive or negative rewards to update the policy. Model-based approaches have difficulty modeling dynamic environments and users, while model-free approaches solve this problem. Model-free methods typically involve learning the value function or learning the policy. Tapus et al. [55] propose an adaptive system that learns user personality through three main parameters: distance of interaction, movement speed, and verbal communication (volume and speed of speech). They used a Policy Gradient Reinforcement Learning algorithm to optimize these three parameters according to the personality of the user in order to maximize user performance. Q-learning is also a widely used algorithm regarding HRI. Hemminahaus and Kopp [25] use Q-learning to adjust the robot's level of assistance to help the human complete tasks. Park et al. [43] use Q-learning to generate personalized robot's action to improve children's engagement and outcomes. Maroto-Gómez et al. [15] propose a Q-learning system with dynamic learning rates to produce personalize robot's behaviors based on the people around it. Standard Q-learning faces the challenge of learning massive amounts of states. Castro-González et al. [14] employ Object Q-learning [33] to reduce the number of states that the agent needs to learn. Deep neural networks with the generalization ability are also a popular approach to solve this challenge. In [45], the robot used the deep policy network to learn social behaviors. However, there is no guarantee of convergence when applying neural networks as the function approximator [6][33].

Interactive Reinforcement Learning (IRL) is an extension of traditional Reinforcement Learning where the learning process is augmented by interactions with a human teacher or advisor. Instead of relying solely on rewards from the environment, the agent can receive guidance, feedback, or even corrective actions from the humans, potentially expediting the learning process and aligning the agent's behavior with human expecta-

tions. Senft et al.[52] present a supervised Reinforcement Learning framework to learn personalized behaviors from expert guidance. Tsiakas et al.[57] propose an IRL framework that combines explicit feedback with implicit human-generated feedback. Such approaches allow the robot to adapt flexibly to different environments and situations because of facilitating human-robot interactions. Humans can provide more detailed information about the environment and the task. However, this collaborative form of learning requires human input and supervision. The need for human involvement can also be inefficient if the human guidance and feedback are not consistent or if the robot demands to be constantly supervised and corrected. The collaborator's guidance and feedback may also be biased toward a particular outcome. This thesis, instead, aims at adaptation without human demonstration.

3.5 Summary

User adaptability plays an essential role in developing socially acceptable service robots. There are various ways to implement adaptive robots. We reviewed user adaptability in service robots, which is the main theme of this thesis and essential for building socially acceptable robots. Human capabilities are a more objective perspective than personal preferences. Nevertheless, limited research involves online adaptation of human capabilities in high-level task planning, which is the central focus of this thesis.

User models serve as knowledge bases, enabling robots to optimize their behavior, and ensuring alignment with individual users' preferences and intentions. Inspired by existing research, we divide the persistence of user models into three types: static, dynamic and implicit. Static models use information collected during or before interactions and remain fixed throughout, while dynamic models update the knowledge related to users over time. Implicit models do not require explicit user models and instead use direct feedback obtained during the interaction to optimize the robot's policy. We review three decision-making methods: rule-based systems, supervised learning, and Reinforcement Learning. The adaptability of intelligent service robots is crucial to their successful deployment in real-world environments, which are inherently uncertain. Although traditional modeling methods based on domain knowledge and questionnaires can provide stable adaptive policies in controlled environments, dynamic user models are more effective in long-term adaptive and multi-agent collaboration tasks. Learning is a possible solution to dynamic environments where responses to all different situations cannot be pre-programmed. Our work includes explicit, predefined domain knowledge to clarify the correlation between human capabilities and robot

actions. However, we apply online Reinforcement Learning guided by independent capability estimation in our framework to support the agent in continuously improving its policy to achieve dynamic user adaptability.

Robust adaptability, quick adaptation to new users, task generalization, acceptance of guidance from non-experts, and adaptive multi-agent interactions are central challenges in this field. Overall, the success of service robots in the home and public domains hinges on their ability to adapt to diverse user needs and preferences.

Chapter 4

Human Capability Estimation through Reinforcement Learning

4.1 Introduction

In this chapter, we explore the challenges robots face when estimating human capabilities and then use that knowledge to enhance user adaptability in human-robot collaboration tasks. We introduce a framework for estimating human capabilities that combines Bayesian inference and online Reinforcement Learning. Instead of focusing on the causes behind human capabilities, our approach prioritizes the robot's perspective. One of our goals is to determine the specific human capabilities required for each collaborative action that the robot can perform. Let us review the scenario of a robot providing a guided tour. The robot can drive at high speeds, requiring the person to keep up at a similar pace. However, due to factors like age, injuries, or disabilities, the person may be unable to move quickly. Rather than evaluating the physical reasons (e.g., diseases and pathology) behind these limitations, our framework focuses on the preconditions of the robot's actions. If the robot gathers sufficient evidence indicating that the person cannot benefit from a certain action, it will disable that action accordingly.

The capability estimation framework we propose introduces an additional abstraction layer, orthogonal from the observable state space, and not directly perceivable by the robot. Capabilities are inherent properties of individuals, making them latent variables. We introduce a Bayesian method for capability estimation, which gathers evidence through RL, avoiding the demands for the more complex and computationally intensive model of POMDPs. Based on the previous example of the guided tour, the RL agent may receive a low reward when it drives fast, as it leaves the person behind. The

agent connects this information to the observable state space and learns that driving fast is not a suitable choice in the given location with that specific person. However, to generalize this knowledge and understand that driving fast is unsuitable for this person, the agent would need to receive low rewards across various locations consistently. With the incorporation of capability estimation, the latent capability of the person to move fast can be inferred as negative at an early stage. Once sufficient evidence has been accumulated, the driving-fast action can be universally deactivated, regardless of the location. Such a system makes informed decisions based on the person's capabilities, enhancing the robot's efficiency and adaptability in human-robot collaboration tasks.

4.2 Problem Definition

We build on the common task model of a MDP denoted as $D = \langle S, A, P_a, R_a, \gamma \rangle$, where S is a set of states, A is a set of actions, P_a is the transition function, R_a is the reward function, and γ is the discount factor. We augment this model with the collaborator's capabilities as preconditions to robot actions. For instance, in the tour guide example, the robot's action `drive_fast` depends on the collaborator's capability to `move_fast`.

We define $\mathbb{L} = \{\lambda_1, \lambda_2, \cdots, \lambda_n\}$ as the label set of n human capabilities. In the tour guide example, $\mathbb{L} = \{\texttt{move\_fast}\}$. For each capability λ_i, we define a binary random variable $c_i \in \{0, 1\}$ to indicate the capability presence, where $c_i = 1$ represents the human has the corresponding capability λ_i, and $c_i = 0$ otherwise. Thus, a specific set of human capabilities can be represented as an n-dimensional binary vector $\mathbf{c} = \{c_1, c_2, \cdots, c_n\} = \{0, 1\}^n$, In the tour guide example, we introduced one human capability `move_fast`, hence, $\mathbf{c} = \{1\}$ represents the capability set where the collaborator can move fast, and $\mathbf{c} = \{0\}$ vice versa. We refer to such a binary vector as a capability combination. For n capabilities, there are up to 2^n capability combinations.

We define for each action the subset of the capabilities that are required for that action, and denote it with $C(a)$. In the tour guide example, $C(\texttt{drive\_fast}) = \{\texttt{move\_fast}\}$. At each time step t, an action a is considered *enabled* if all capabilities in $C(a)$ are present. Mathematically, the set of enabled actions at time t is defined as $A_t = \{a \in A \mid \forall c_i \in C(a); c_i = 1\}$.

We define the policy π of the RL agent as $\pi(a \mid s, \mathbf{c})$, which is affected by both state and capabilities, represented by a Bayesian network in the Figure 4.1. The primary objective of the RL agent is to calculate an optimal policy, denoted as $\pi^*(a \mid s, \mathbf{c})$ which maximizes the expected return $G = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t r_{t+1}\right]$ in the given task, where r_t is the immediate reward at time step t, extracted from R_a. The policy is constrained

to select actions exclusively from the set of enabled actions, which corresponds to the domain of the random variable $\mathcal{A}$ in $\pi(\mathcal{A} = a \mid s)$ is A_t.

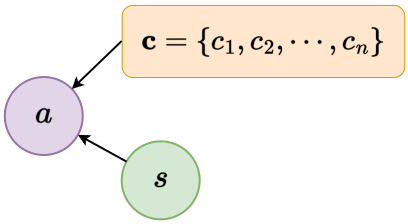


Fig. 4.1 The Bayesian network of the policy (π) affected by human capabilities.

The explicit estimation of the probability of human capabilities offers the robot the ability to accumulate evidence over time. When the probability of a particular capability $P(c_i \mid \texttt{evidence})$ is reliably low, the robot can assume the absence of capability c_i and disable the corresponding actions according to $C(a)$, irrespective of the current state partially or entirely. Consequently, an essential aspect of our framework lies in accurately estimating human capabilities. In the subsequent section, we will provide a detailed description of our approach to address this crucial aspect.

4.3 Capability Estimation Strategies

We propose indirectly estimating the probability of human capabilities through the underlying continuous RL process. In this section, we introduce a Bayesian framework for capability estimation and will explore the implementation details and offline training in subsequent sections.

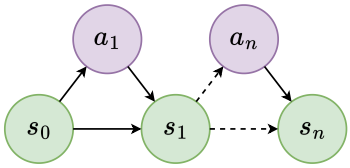


Fig. 4.2 The Bayesian network of MDP

The Bayesian network shows the correlation between states and actions within the standard MDP model, as shown in Figure 4.2. The vector of capabilities is considered as a latent variable, thus it cannot be observed online directly. The training and testing process of the capability estimation model is as follows: The RL agent is pre-trained (ct. Section 4.5.2) under known human capabilities firstly, converging to the optimal policy $\pi^*(a \mid s, \mathbf{c})$. Then, the robot is deployed with a new user and expected to adapt

to the user's actual capabilities as quickly as possible. The underlying RL process selects actions while improving the policy and thereby generating a sequence of states, actions, and rewards. We define the trajectory τ as the feature vector, comprising a sequence of tuple $(s_t, a_t, s_{t+1}, r_{t+1})$, where s_t denotes the state at time t, a_t denotes the action performed by the agent at time t, s_{t+1} denotes the state at time $t+1$, the r_t denotes the immediate reward at time $t+1$. The trajectory does not contain an entire episode, and the entire task is possibly infinite, but we consider it in batches of length l. Given such a trajectory as the evidence, we can obtain estimates of human capabilities based on Bayes' theorem shown in Equation (4.1), where $P_r(\mathbf{c})$ represents the prior probability that expresses our beliefs about the capabilities $\mathbf{c}$ before we see any evidence:

$$P(\mathbf{c} \mid \tau) = \frac{P(\tau \mid \mathbf{c}) P_r(\mathbf{c})}{P(\tau)}. \tag{4.1}$$

Although τ contains all observable information in the standard MDP, not all variables within this information are necessary for capability estimation. To explore the influence and correlations among these variables on capability estimation, we propose the following three estimation strategies: EST-Action, EST-State-Policy, and EST-Policy.

EST-Action: We define the trajectory τ_a to be a sequence of the historical actions taken:

$$\tau_a = \{a_1, a_2, ..., a_l\}. \tag{4.2}$$

We disregard connections between actions caused by the state transition function P_a and assume that all actions in trajectory τ_a are conditionally independent. The Bayesian network depicting the correlation between the capability vector and actions is shown in Figure 4.3. Therefore, Equation 4.1 derives to Equation (4.3) according to the Naive Bayesian:

$$P(\mathbf{c} \mid \tau_a) = \frac{\prod_{i=1}^{l} P(a_i \mid \mathbf{c}) P_r(\mathbf{c})}{\sum_{\mathbf{k} \in 2^{\mathbf{c}}} \prod_{i=1}^{l} P(a_i \mid \mathbf{k}) P_r(\mathbf{k})}. \tag{4.3}$$

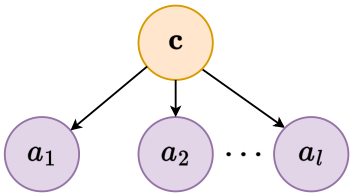


Fig. 4.3 The Bayesian network of EST-Action

The likelihood $P(a_i \mid \mathbf{c})$ in Equation (4.3) is the action distribution given capabilities $\mathbf{c}$. To obtain that, we use the sampling method over a number of episodes in which the RL agent executed the optimal policy $\pi^*(a \mid s, \mathbf{c})$. We assume that the action space is discrete, thus, by setting action counters to track the occurrences of each action, we can compute the approximate action distribution based on Equation (4.4). The parameter w determines the number of episodes of the sampling process. The Behavior Policy of the value-based RL agent can adopt the Softmax method, resulting in the opportunity to be performed for all actions. Smoothing algorithms can also be applied to Equation (4.4) to avoid zero counts in computing, ensuring a more robust estimation of action probabilities:

$$P(a_i \mid \mathbf{c}) = \frac{\sum_w count(a_i) \sim \pi^*_{\mathbf{c}}}{\sum_w \sum_{a \in A} count(a) \sim \pi^*_{\mathbf{c}}}. \tag{4.4}$$

EST-State-Policy: We define the trajectory τ_π includes the sequence of tuple consisting of states and actions experienced by the agent:

$$\tau_\pi = \{(s_0, a_1), (s_1, a_2), ..., (s_{l-1}, a_l)\}. \tag{4.5}$$

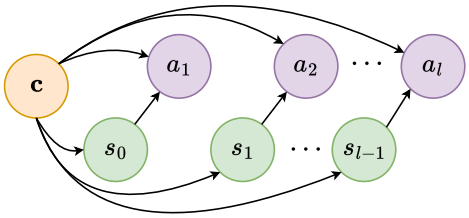


Fig. 4.4 The Bayesian network of EST-State-Policy

The Bayesian network presented in Figure 4.4 illustrates correlations among random variables in this strategy. We disregard the effects of the state transition function P_a, in other words, the states are considered conditionally independent. The capability vector acts on every state and action, except for the final state. Therefore, Equation 4.1 derives to Equation (4.6) according to the Naive Bayesian:

$$P(\mathbf{c} \mid \tau_\pi) = \frac{\prod_{i=1}^{l} \pi^*(a_i \mid s_{i-1}, \mathbf{c}) P(s_{i-1} \mid \mathbf{c}) P_r(\mathbf{c})}{\sum_{k \in 2^{\mathbf{c}}} \prod_{i=1}^{l} \pi^*(a_i \mid s_{i-1}, \mathbf{k}) P(s_{i-1} \mid \mathbf{k}) P_r(\mathbf{k})}. \tag{4.6}$$

Similar to Equation (4.4), Equation (4.7) shows that the approximate state distribution can be obtained by tracking the occurrences of each state given the optimal policy $\pi^*(a \mid s, \mathbf{c})$. Thus, unlike EST-Action, which requires a discrete action space,

EST-State-Policy is applicable when dealing with discrete state space:

$$P(s_i \mid \mathbf{c}) = \frac{\sum_w count(s_i) \sim \pi^*_{\mathbf{c}}}{\sum_w \sum_{s \in S} count(s) \sim \pi^*_{\mathbf{c}}}. \tag{4.7}$$

EST-Policy: The trajectory τ_π used in EST-State-Policy is still employed in this strategy. However, we take into account the effect of state transition function P_a, and capabilities are considered only to affect actions. The Bayesian network representing this strategy is shown in Figure 4.5. Consequently, the likelihood in Equation 4.1, for a given policy π, can be expressed as follows:

$$\begin{aligned} P(\tau_\pi \mid \mathbf{c}) =& P(s_0)\pi^*(a_1 \mid s_0, \mathbf{c})P_a(s_1 \mid s_0, a_1)\pi^*(a_2 \mid s_1, \mathbf{c}) \cdots \\ & P_a(s_{l-1} \mid s_{l-2}, a_{l-1})\pi^*(a_l \mid s_{l-1}, \mathbf{c}). \end{aligned} \tag{4.8}$$

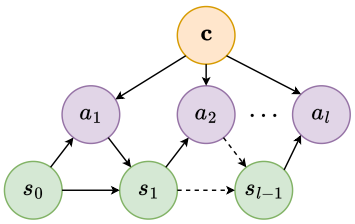


Fig. 4.5 The Bayesian network of EST-Policy

The probability of the trajectory can be factored into two parts, one that depends on the capabilities, and one that does not:

$$P(\tau_\pi \mid \mathbf{c}) = P(s_0) \prod_{i=1}^{l-1} P_a(s_i \mid s_{i-1}, a_i) \prod_{i=1}^{l} \pi^*(a_i \mid s_{i-1}, \mathbf{c}). \tag{4.9}$$

The first two terms in the right-hand side of Equation 4.9, which do not depend on the capabilities, appear in both the numerator and the denominator of Equation 4.1 (that is, in both $P(\tau_\pi \mid \mathbf{c})$ and $P(\tau_\pi)$) and therefore cancel each other out. The estimation of the capabilities can then be obtained from the optimal capability-conditioned policy and a trajectory through the Bayesian update:

$$P(\mathbf{c} \mid \tau_\pi) = \frac{\prod_{i=1}^{l} \pi^*(a_i \mid s_{i-1}, \mathbf{c}) P_r(\mathbf{c})}{\sum_{\mathbf{k} \in 2^{\mathbf{c}}} \prod_{i=1}^{l} \pi^*(a_i \mid s_{i-1}, \mathbf{k}) P_r(\mathbf{k})}. \tag{4.10}$$

In EST-Policy, we circumvent the discretization required for the action and state spaces in the earlier EST-Action and EST-State-Policy strategies. This enhancement provides EST-Policy with a broader scope of applicability.

4.4 Capability Belief

In the previous section, we presented estimation strategies for the capability combination $\mathbf{c}$. The resulting multi-label learning of Equation (4.3), (4.6) and Equation (4.10) can be simplified in case of dependency between capabilities, but in this thesis, we apply the Label Powerset method, thus, consider the full set of $|2^{\mathbf{c}}|$ combinations.

By merging the joint probabilities of all capability combinations from Equations (4.3), (4.6) and (4.10), we can compute the marginal probabilities for each capability c_i's presence given the trajectory τ, denoted with $P(c_i = 1 \mid \tau)$, which reflects the degree of belief in the capability c_i. This distribution can either be the direct output from the estimation or be transformed into a binary outcome using the following two policies:

Deterministic Policy: we define a set of thresholds $\mathbf{d} = \{d_1, d_2, \cdots, d_n\}$, where each threshold d_i corresponds the capability c_i. Capabilities with a belief value greater than or equal to the corresponding threshold are labeled as positive, while those falling below are labeled as negative, as shown in Equation (4.11). Recall from Section 4.2 where we defined $C(a)$ as the capabilities required to execute the action a. By applying this deterministic policy, the set of enabled actions at time t can be further defined as $A_t = \{a \in A \mid \forall c_i \in C(a); P(c_i = 1 \mid \tau) >= d_i\}$:

$$\forall c_i \in \mathbf{c},\ c_i = \begin{cases} 1 & P(c_i = 1 \mid \tau) >= d_i \\ 0 & \text{otherwise.} \end{cases} \tag{4.11}$$

.

Stochastic Policy: The presence of a capability follows its probability distribution, as shown in Equation (4.12), where the function `rand()` returns a uniform random value between 0 and 1. By applying this stochastic policy, the set of enabled actions at time t can be further defined as $A_t = \{a \in A \mid \forall c_i \in C(a); c_i \sim P(c_i = 1 \mid \tau)\}$:

$$\forall c_i \in \mathbf{c},\ c_i = \begin{cases} 1 & P(c_i = 1 \mid \tau) >= \texttt{rand}() \\ 0 & \text{otherwise.} \end{cases} \tag{4.12}$$

4.5 Implementation and Deployment

We introduced the problem definition concerning estimating human capability probabilities within the RL in Section 4.2. It was followed by the proposition of estimation strategies in the context of RL in Section 4.3. Subsequently, in Section 4.4, we presented the output beliefs of capability estimation. In this section, we introduce the

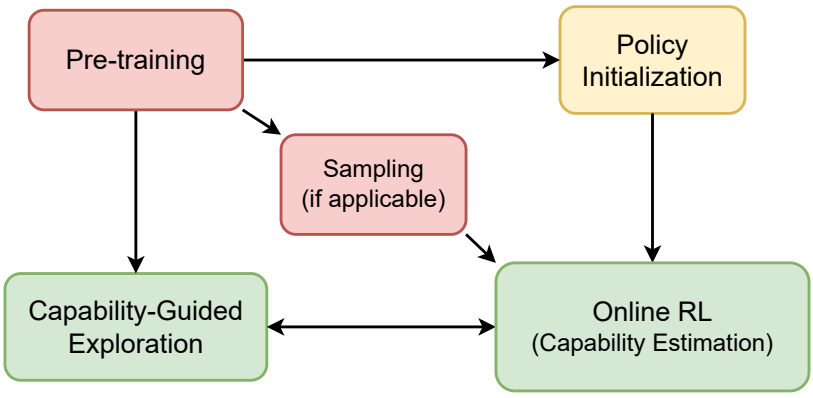


Fig. 4.6 The main components of our framework

implementation and deployment aspects of our capability estimation framework. Figure 4.6 summarizes the critical aspects of our framework and the data flow among them:

- Pre-training of the capability-conditioned policy $\pi^*(a \mid s, \mathbf{c})$ under known capability combinations.
- Estimating state distribution $S \sim$ or action distribution $A \sim$ by sampling if applicable.
- Initialization of the value function using the pre-trained policies.
- Online RL with a new user, while continually estimating capabilities using EST-Action, EST-State-Policy, or EST-Policy.
- Taking advantage of the capability estimation for exploration.

The red highlights in Figure 4.6 represent the training process. The policy initialization highlighted in yellow represents the initialization of online learning. The green highlights procedures of online learning with capability estimation. We will go through each aspect presented above. To provide a clear illustration, we will use the first task from our experimental evaluation as an example and begin with its description.

4.5.1 Robot Navigation Task

In this collaborative navigation task, the robot has to lead a person to a desired location with possible human help. The Gazebo[1] simulation environment is shown in Figure 4.7. The robot starts from a fixed initial location, marked in green, and is asked to

[1]https://gazebosim.org/

lead the person to the goal location in blue. Two red doors provide shortcuts in the environment. Only the collaborator can open doors. We introduce more details of the implementation and evaluations in Section 5.2.

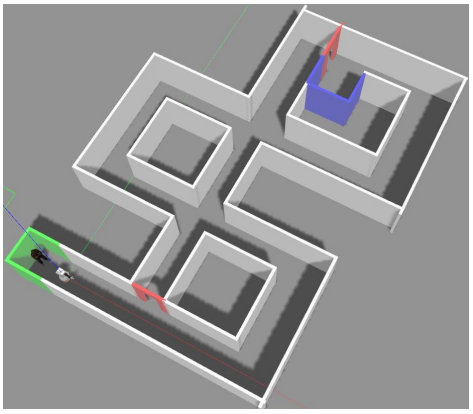


Fig. 4.7 The Gazebo world for the navigation task

The state space is composed of the robot's position, the status $\{\texttt{close}, \texttt{safe}, \texttt{far}\}$ used to represent the distance between the robot and the collaborator, and whether each door is open or closed. The action space consists of nine actions: four navigation actions in the four cardinal directions both driving fast and slow, and the action `open_door` to ask the collaborator to open doors and take the shortcut. Slow driving actions make the robot move 1 unit per step, and fast driving actions result in 2 units. Two human capabilities, $\mathbb{L} = \{\texttt{c\_fast}, \texttt{c\_open}\}$, are considered, to enable the fast navigation actions and the `open_door` action. The human model always catches up to the robot if $\texttt{c\_fast} = 1$, otherwise 1 unit per step. Significant time may be saved by passing doors and taking shortcuts. Every step will result in a negative reward of -1. Reaching the goal will bring a reward of 50. When the distance status is `far` (distance exceeds 1 unit), the agent will be assigned a negative reward of -50.

To make the environment realistic, we introduce the following features: The robot's perceived human-robot distance is affected by Gaussian noise ($\mu = 0, \sigma = 0.05$); Each door may be open at the start of every episode with probability 0.1; The human model makes slow steps even if $\texttt{c\_fast} = 1$ with probability 0.05.

4.5.2 Pre-training

All estimation strategies we proposed in Section 4.3 require optimal policies $\pi^*(a \mid s, \mathbf{c})$, which can be trained offline under a sufficiently representative and known set of capabilities. This training can be entirely model-free, executing the task in the real

world with a group of people with different known capabilities. However, such an effort would be considerable and, in many cases, unrealistic. In this thesis, we rely on simulations modeling the behavior of people with different capabilities, and on which the training to learn the optimal policy $\pi^*(a \mid s, \mathbf{c})$ can be carried out offline. The simulation will inevitably be different from real users. However, the capability estimate can be made tolerant to uncertainty by adjusting the set of thresholds $\mathbf{d}$ (ct. Section 4.4).

In this navigation task, offline pre-training results in five optimal policies. Four of these pre-trained policies are for deterministic capabilities, corresponding for each combination of $\texttt{c\_open} = \{0, 1\} \times \texttt{c\_fast} = \{0, 1\}$. And one remaining pre-trained policy is trained with random capabilities, denoted with $\pi^*(a \mid s, random)$, which is mainly used as an initial policy of online learning we will introduce in Section 4.5.4. When training this policy for random capabilities in the simulator, the human model stimulates human capabilities that are randomly generated at the start of each episode. These pre-trained policies and their associated value functions serve multiple purposes: (i) they support the sampling process to estimate state distribution and action distribution, which we will introduce in Section 4.5.3; (ii) the agent can use them as the initial value function of online learning, which we will introduce in Section 4.5.4; (iii) they support the capability-guided exploration, which we will introduce in Section 4.5.6.

Chunking Storage

Since both the state space and the action space in this task are discrete, we implemented EST-Action, EST-State-Policy and EST-Policy by using the Q-learning algorithm. We also implemented EST-Policy by using the DQN algorithm. Because the pre-training process results in $|2^{\mathbf{c}}| + 1$ policies, and each policy contains a Q-table of size $|S| \times |A|$. The storage and memory demands of these policies are substantial, presenting challenges to the program running. We developed a storage solution based on Zarr[2]. Zarr is an open-source format that offers features such as chunking and compressed storage, and it seamlessly integrates with popular data science libraries like Numpy[3].

"Chunking" is a term for breaking up and organizing big data within a multi-dimensional array. In Zarr, data is stored in a hierarchical structure where each array comprises chunks. Each chunk is a smaller sub-array of the larger data array. The

[2]https://zarr.readthedocs.io/
[3]https://numpy.org/

chunking strategy involves dividing the array into smaller chunks and includes the following key benefits:

(i) **Partial Loading**: With chunking, it becomes possible to read only specific parts of a large array without loading the entire array into memory. It is essential when working with large Q-tables that do not fit into memory.

(ii) **Efficient Updating**: Chunking allows us to write data in smaller units, which can improve I/O performance. Instead of saving the entire large array at once, only the relevant chunks are updated, reducing the amount of data transferred and improving saving times.

(iii) **Compression**: Zarr supports chunk-level compression, where each chunk of data can be individually compressed before being stored. It can significantly reduce storage requirements, especially for datasets with repetitive or compressible patterns.

After examining storage protocols provided by Zarr, we selected the one based on Lightning Memory-Mapped Database (LMDB)[4] due to its exceptional stability. By implementing our storage solution, we have achieved a significant reduction of at least 50% in the hard disk and memory resources required. This solution allows our framework to run on platforms with limited resources, such as the robot's internal computers.

One-hot Encoding

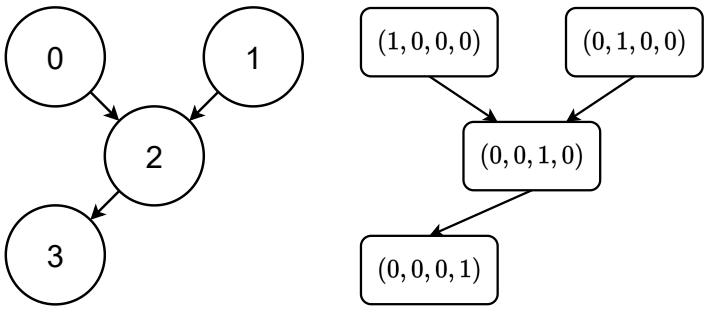


Fig. 4.8 An example of one-hot encoding: (a) The directed graph with nodes in the label style. (b) The directed graph with nodes in one-hot encoding.

[4]http://www.lmdb.tech/doc/

For DQN, we employ One-hot encoding to transform the discrete state values into a binary vector representation because some state values lack a meaningful partial order. In one-hot encoding, each discrete value is represented by a binary vector where only one element is "hot" (1) while the rest are "cold" (0). For instance, in the case of the serial number of nodes in Figure 4.8(a), there is no inherent positional relationship among these numeric labels. This encoding results, shown in Figure 4.8(b), extend the discrete features to the Euclidean space, enabling more meaningful distance calculations between features. Our DQN agent is implemented based on Stable-Baselines3, a set of reliable implementations of RL algorithms [46].

4.5.3 Sampling

For the strategies EST-Action and EST-State-Policy, it is required to employ sampling to obtain the approximate distributions for actions $A \sim$ or states $S \sim$. Below is how the sampling process works:

1. For every capability combination $\mathbf{c} \in 2^{\mathbf{c}}$, we begin by initializing the RL agent with the pre-trained optimal policy corresponding to the capabilities $\mathbf{c}$, that is $\pi^*(a \mid s, \mathbf{c})$.

2. This RL agent then executes w episodes without any exploration.

3. After w episodes, if our strategy is EST-Action, we compute the approximate action distributions for all action $a \in A$ for the given capability combination $\mathbf{c}$, applying Equation (4.4).

4. Conversely, if our strategy is EST-State-Policy, we compute the approximate state distributions for all state $s \in S$ for the given capability combination $\mathbf{c}$, applying Equation (4.7).

4.5.4 Policy Initialization

The underlying online learning is generalized policy iteration, and its convergence is not affected by the initial value function (in MDPs). However, it is possible to expedite the learning process considerably through a favorable initialization. The pre-training phase offers additional advantages by providing such a beneficial initialization, thereby boosting the learning rate.

The robot has a set of $|2^{\mathbf{c}}|+1$ value functions to choose from and use while adapting to the particular user. The policy trained with a user has all capabilities, that is,

$\pi^*(a \mid s, \{1, 1, \cdots, 1\})$ may seem like a natural choice since all actions are enabled in this policy. However, committing to any initial capability combination has a risk of overfitting. Therefore, a competitive option for initialization could be the policy pre-trained over random capabilities, that is, $\pi^*(a \mid s, random)$.

4.5.5 Estimation Algorithm

Online capability estimation is a direct application of EST-Action, EST-State-Policy and EST-Policy. The Algorithm 2 outlines the procedure for applying these three strategies. The estimation process begins with a prior distribution $P_r(\mathbf{c})$. The robot collects l actions or state-action pairs, and then updates the probabilities of the capabilities. The `agent_step()` function at Line 4 returns the most recent action and indicates whether it resulted from exploration, allowing for its exclusion from capability estimation. Exploratory actions do not follow the learned policy and can distort the capability estimates.

Algorithm 2: Human Capability Estimation

Input: $P_r(\mathbf{c})$: The prior probabilities of capabilities.
l: The batch length.

```
τ ⟵ [];
j ⟵ 0;
while learning is running do
    s, a, e ⟵ agent_step();
    if e is false then
        if EST-Action then
            Append the action a to τ;
        else if EST-State-Policy or EST-Policy then
            Append the state-action pair (s, a) to τ;
        end
        j ⟵ j + 1;
    end
    if j mod l = 0 then
        if EST-Action then
            Update capability estimates according to Equation (4.3);
        else if EST-State-Policy then
            Update capability estimates according to Equation (4.6);
        else if EST-Policy then
            Update capability estimates according to Equation (4.10);
        end
        P_r(c) ⟵ current estimate;
        τ ⟵ [] ;
    end
end
```

4.5.6 Capability-guided Exploration

To ensure the convergence of RL, the inclusion of random exploration is mandatory. Because in complex environments, there might be multiple sub-optimal policies that an RL agent could get stuck in if it only exploits its current knowledge. Random exploration allows the agent to venture into unexplored regions of the state-action space

and discover better behaviors that it might not have encountered through deterministic exploitation.

In principle, the robot starts with the average good pre-trained policy for every capability combination, therefore when sufficiently confident in the presence of the capability, it could switch to the corresponding optimal pre-trained policy. However, the capability-conditioned optimal policy was learned in pre-training, it may not be optimal with the new user at hand. Nonetheless, we propose to use it for exploration advice, limiting purely random exploration. Below, we outline two distinct exploration strategies that are guided by capabilities:

RLC: We introduced the notation $C(a)$ in Section 4.2 to denote the preconditions of the agent's actions. This predefined dependency serves as a guiding principle, aiding the RL agent in more efficient exploration while mitigating the need for entirely random behaviors. At each time step t, an action a is considered *enabled* in the exploration if the estimates of all the capabilities in $C(a)$ are positive. Mathematically, the set of enabled exploring actions at time t is defined as $A_t = \{a \in A \mid \forall c_i \in C(a); c_i = 1\}$, where the capability belief c_i can be extracted by the deterministic or stochastic policy presented in Section 4.4.

RLC-Policy: We introduced Pre-training in Section 4.5.2. The resulting pre-trained optimal policy can further guide the agent's exploration. On the basis of RLC, the agent will explore the optimal action in the pre-trained policy corresponding to the latest capability estimates, given the current state s. Mathematically, the possible optimal exploring action given the state s at time t is defined as $a_t \sim \pi^*(a \mid s, \mathbf{c}_{est})$.

The two above exploration strategies we used are shown in Algorithm 3. The capability-guided exploration should be executed on top of ϵ-greedy this algorithm results in choosing the current optimal action with probability $1 - \epsilon$, the capability-guided action with probability ϵ. We can decay κ over time, so the advice is gradually removed.

Algorithm 3: Capability-guided Exploration

Data: A: The Set of Actions.
π^*: The pre-trained optimal policy.
s: The current state.
$\mathbf{c}_{est}$: The current capability estimates.
κ: The value of κ.
κ_{pre}: The value of κ_{pre}.
`decay`(n): The optional decay function, where n is the number of episode.

Function `CapabilityGuidedExploration`(n)**:**
  $\lambda \longleftarrow$ uniform random value between 0 and 1;
  $\kappa \longleftarrow$ `decay`(n);
  if $\lambda < \kappa$ **then**
    if *RLC* **then**
      $a \longleftarrow$ `random`$(\{a' \in A \mid \forall c_i \in C(a'); c_i = 1\})$;
    else if *RLC-Policy* **then**
      $\lambda_{pre} \longleftarrow$ uniform random value between 0 and 1;
      if $\lambda_{pre} < \kappa_{pre}$ **then**
        $a \longleftarrow$ `random`$(\{a' \in A \mid \forall c_i \in C(a'); c_i = 1\})$;
      else
        $a \sim \pi^*(a \mid s, \mathbf{c}_{est})$;
      end
    end
  else
    $a \longleftarrow$ `random`(A);
  end
  return a;

4.6 Summary

In this chapter, we proposed a human capability estimation framework based on Reinforcement Learning and Bayesian inference. In Section 4.2, we introduced the problem definition of estimating human capabilities in the context of RL. Human capabilities as preconditions for robot actions. In Section 4.3, we introduced three estimation strategies for the capability combination: EST-Action, EST-State-Policy, and EST-Policy. In Section 4.4, we introduced methods for extracting capability beliefs

from continuous distributions. In Section 4.5, we introduced various critical aspects of our proposed framework through the robot navigation experiment, which mainly includes pre-training, sampling, policy initialization, capability estimation algorithm, and two capability-guided exploration approaches: RLC and RLC-Policy. In the next chapter, we will introduce the evaluation methods of our framework, three human-robot collaboration experiments, and experimental results.

Chapter 5

Experimental Evaluation

For all our experiments, we used the PAL Robotics TIAGo[1], both in a Gazebo simulation and the real world. All environment is designed to meet the OpenAI Gym[2] standard, and the program interface is extended to support collaborative tasks where robots need to adapt to human capabilities. This standardized structure ensures that the task environment can be easily utilized with other Reinforcement Learning methods, enabling quick and seamless integration into different algorithms.

5.1 Evaluation Metrics

Our evaluation focuses on the agent's performance in RL and the effectiveness in multi-label learning. For RL performance, we directly compare the episode *Return* and the number of steps in episodes. Evaluating multi-label learning is more complex than single-label learning [65]. Let L denote the set of labels, D denote the set of samples, Z denote the set of true sample pairs, and Y denote the set of predicted sample pairs. We firstly introduce the *Accuracy*, *Precision* and *Recall* presented by Godbole et al. [19], shown by Equations (5.1), (5.2), (5.3). Higher values of *Accuracy*, *Precision*, and *Recall* indicate better performance of the classifier:

$$Accuracy = \frac{1}{|D|}\sum_{i=1}^{|D|}\frac{|Y_i \bigcap Z_i|}{|Y_i \bigcup Z_i|}, \tag{5.1}$$

$$Precision = \frac{1}{|D|}\sum_{i=1}^{|D|}\frac{|Y_i \bigcap Z_i|}{|Z_i|}, \tag{5.2}$$

[1]https://pal-robotics.com/robots/tiago/
[2]https://github.com/openai/gym

$$Recall = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{|Y_i \bigcap Z_i|}{|Y_i|}. \tag{5.3}$$

The *Hamming Loss*, introduced by Schapire and Singer [50], is another metric we used to quantify the performance of multi-label learning tasks. This metric is defined as the fraction of labels incorrectly predicted, on average, across all instances. In other words, it calculates the proportion of label assignments that are different between the predicted labels and the true labels for all instances in the dataset. The Equation of *Hamming Loss* is shown on (5.4), where Δ represents the Kronecker delta function, and this operation is similar to the XOR (exclusive OR) operation in Boolean logic [58]. For *Hamming Loss*, a lower value is better.

$$Hamming\ Loss = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{|Y_i \Delta Z_i|}{|L|}. \tag{5.4}$$

5.2 Robot Navigation Task

The configuration of the robot navigation task was partially introduced in Section 4.5.1. This section will continue introducing more details of the implementation, experimental results, and analysis.

5.2.1 Human Following

The simulated collaborator must have a navigation functionality similar to that of an actual human to follow the guiding robot. Initially, we attempted to deploy two independent TIAGo robots in Gazebo, each equipped with autonomous navigation and obstacle avoidance functions. One robot served as a guide, while the other acted as a collaborator. However, this approach yielded unsatisfactory results, primarily due to the following reasons:

(i) The collaborator is usually close to the guiding robot. Thus, autonomous navigation systems often misinterpret each other as obstacles, leading to navigation stuck or unnecessary detours.

(ii) Given the relatively tiny distance between the two robots, if the guiding robot suddenly stops, the collaborator may struggle to stop promptly. This challenge arises from the simulator's acceleration and friction parameter limitations.

(iii) Positioning error is allowed in the navigation stack of both robots, causing both the guiding robot and the collaborator to deviate from their intended positions.

(iv) TIAGo has no omnidirectional base, which restricts the ability of both the robot and the collaborator to make precise attitude adjustments. This limitation further damaged the robot model's suitability for acting as a collaborator, as its navigation behavior does not possess the flexibility that an actual human does.

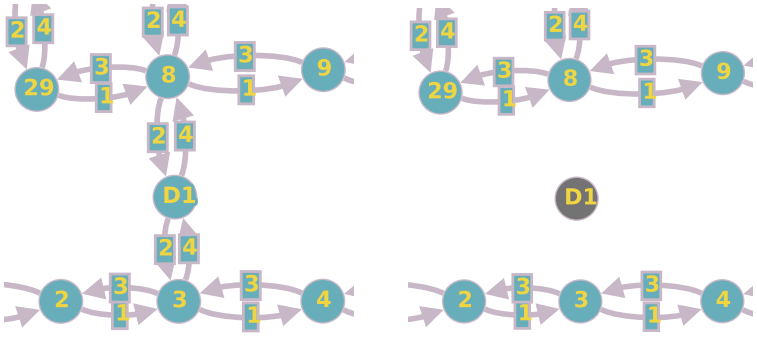


Fig. 5.1 Partial directed graph of the navigation task. (a) The door `D1` is open. (b) The door `D1` is closed.

To resolve the above navigation issues, we represented all paths of the entire environment by a directed graph. Part of this graph is shown in the Figure 5.1. The continuous Cartesian coordinates in the simulator are reduced to a series of waypoints, represented by numerical nodes in the graph. Both the collaborator and the guiding robot in the simulator move between these waypoints. The weight of the edges in the graph is used to indicate the direction of movement between waypoints. Specifically, `1` indicates moving right, `3` indicates moving left, `2` indicates moving down, and `4` indicates moving up. Doors are represented by nodes starting with the letter `D`. Figure 5.1(b) shows a closed door, and its associated node is deactivated, preventing the robot and collaborator from passing through. The robot can request the collaborator to open doors only when the robot is located in a neighboring node of the door node. This simplified representation allows for more efficient modeling and smooth navigation within the simulator.

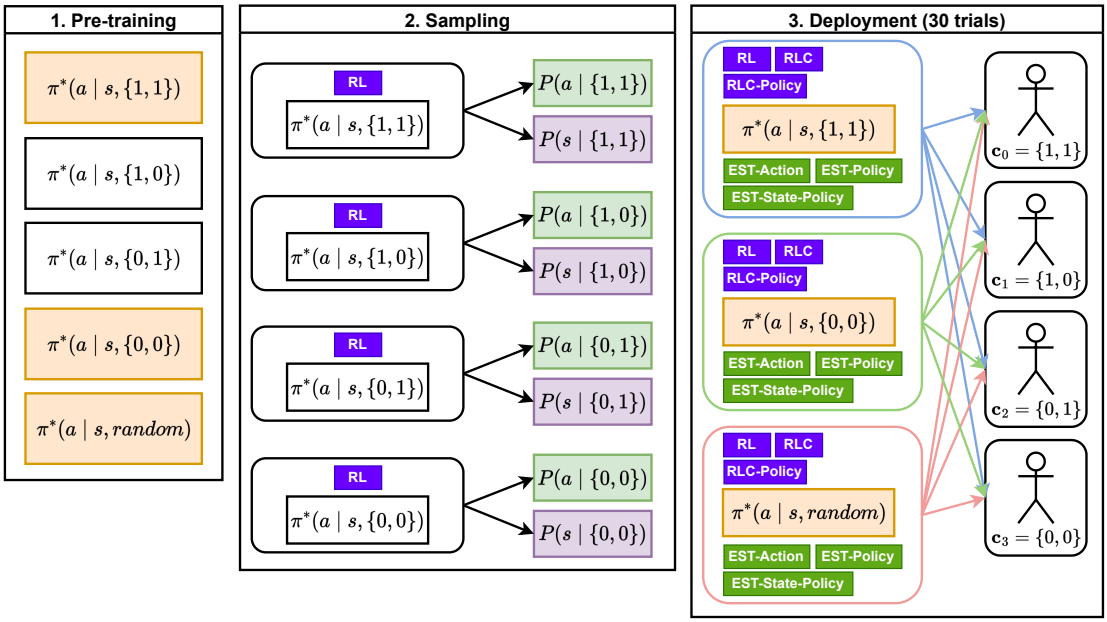


Fig. 5.2 The experimental setup of the navigation task (Q-learning)

5.2.2 Experimental Setup

Q-learning Implementation

The experiment started with pre-training, resulting in $|2^{\mathbf{c}}| = 4$ pre-trained policies and one extra pre-trained policy for random capabilities. The pre-trained policies $\pi^*(a \mid s, \{1,1\})$, $\pi^*(a \mid s, \{0,0\})$ and $\pi^*(a \mid s, random)$ serve as the initial policies. For each initial policy, the RL agents were deployed separately with collaborators of all $|2^{\mathbf{c}}| = 4$ capability combinations. We tested all combinations of {EST-Action, EST-State-Policy, EST-Policy} $\times$ {RL(normal), RLC, RLC-Policy} separately in the deployment. Consequently, we had $4 \times 3 \times 3 \times 3 = 108$ deployment configurations. Figure 5.2 shows the overview of the experimental setup.

In the pre-training, the Q-learning agents were trained over 800,000 episodes for each capability combination. The learning rate $\alpha = 0.1$, the discount factor $\gamma = 0.95$. The maximum number of steps in a single episode was 500. The prior probability of the combination $\{\texttt{c\_fast} = 1, \texttt{c\_open} = 1\}$ was 0.8, and for the other combinations was 0.06.

In the sampling, the Q-learning agents ran 500,000 episodes for each capability combination. The Softmax temperature $\tau = 0.01$. The smoothing factor of Laplace smoothing was 1.

In the deployment, the Q-learning agents ran 15,000 episodes for each configuration. The ϵ of ϵ-greedy decayed linearly from 0.5 to 0.1. The learning rate $\alpha = 0.3$, the discount factor $\gamma = 0.99$. The maximum number of steps in a single episode was 500. The $\kappa = 0.5$ and $\kappa_{pre} = 0.5$. The capability thresholds $\mathbf{d} = \{0.5\}^2$. The Softmax temperature $\tau = 0.01$. The batch size of estimating trajectory $l = 4$.

DQN Implementation

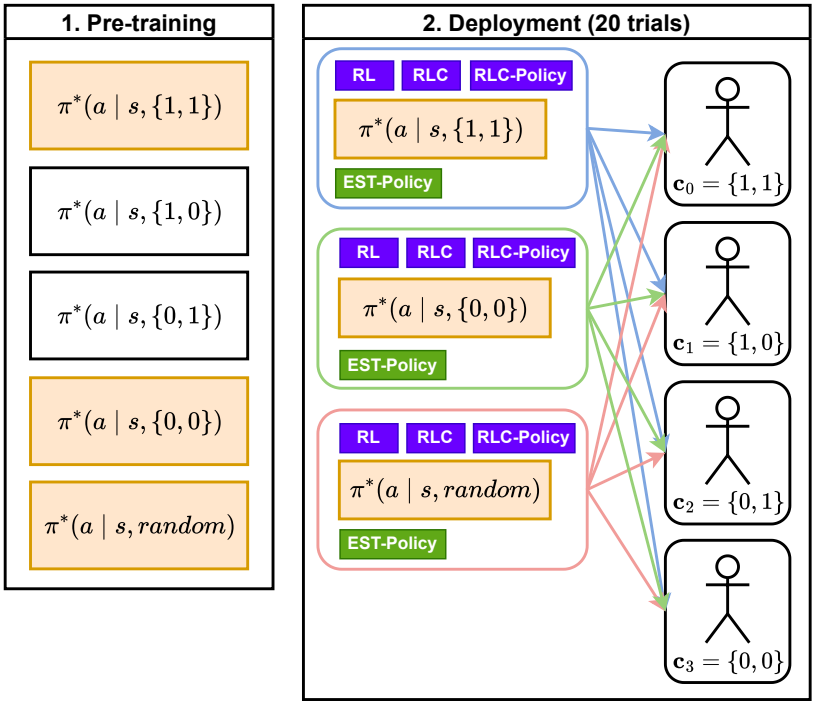


Fig. 5.3 The experimental setup of the navigation task (DQN)

The experiment started with pre-training, resulting in $|2^{\mathbf{c}}| = 4$ pre-trained policies and one extra pre-trained policy for random capabilities. The pre-trained policies $\pi^*(a \mid s, \{1,1\})$, $\pi^*(a \mid s, \{0,0\})$ and $\pi^*(a \mid s, random)$ serve as the initial policies. For each initial policy, the RL agents were deployed separately with collaborators of all $|2^{\mathbf{c}}| = 4$ capability combinations. We tested all combinations of {EST-Policy} $\times$ {RL(normal), RLC, RLC-Policy} separately in the deployment. Consequently, we had $4 \times 3 \times 1 \times 3 = 36$ deployment configurations. Figure 5.3 shows the overview of the experimental setup.

In the pre-training, the DQN agents were trained over 2,000,000 steps for each capability combination. The learning rate $\alpha = 0.001$, the discount factor $\gamma = 0.99$. The maximum number of steps in a single episode was 500. The prior probability of

the combination $\{\texttt{c\_fast} = 1, \texttt{c\_open} = 1\}$ was 0.8, and for the other combinations was 0.06. The batch size of learning was 2048. The ϵ initially started at 0.8 and began to decay. Once the training progress reached 85%, ϵ was reduced to 0 and remained fixed at this value. The policy network consisted of three layers, each with a size of 64.

In the deployment, the DQN agents ran 350,000 steps for each configuration. The ϵ initially started at 0.8 and began to decay. Once the training progress reached 85%, ϵ was reduced to 0 and remained fixed at this value. The $\kappa = 0.5$ and $\kappa_{pre} = 0.5$. The capability thresholds $\mathbf{d} = \{0.5\}^2$. The Softmax temperature $\tau = 0.0001$. The batch size of estimating trajectory $l = 4$.

5.2.3 Estimation Performance Evaluation

In this subsection, we first evaluated the effectiveness of our estimation strategies: EST-Action, EST-State-Policy, and EST-Policy with Q-learning implementation, using *Accuracy*, *Precision* , *Recall* and *Hamming Loss*. The figures of Q-learning implementation displayed in this subsection show the average result across configurations in the Q-learning deployment, accompanied by 95% confidence intervals, calculated based on 30 trials.

Figure 5.4, Figure 5.5 and Figure 5.6 show *Precision* and *Recall* of the three estimation strategies: EST-Action, EST-State-Policy and EST-Policy. The x-axis across these figures represents the learning progression, segmented into 20 intervals. Specifically, $x = 0$ denotes the initial 5% of the learning process, while $x = 19$ corresponds to 100% completion.

The results are all good, and the fluctuations shown in these figures are minimal. We first focus on the performance gaps between different estimation strategies, examining the overall differences among Figure 5.4, Figure 5.5 and Figure 5.6. It can be seen that when the estimated strategy is EST-Action, and the initial policy is $\pi^*(a \mid s, \{0, 0\})$, the value of *Recall* is slightly lower in the beginning compared with other configurations, which is because EST-Action only uses the sequence of actions to estimate, but the agent's policy has not yet converged to the optimal policy in the initial stage. This non-optimal and cautious policy will further cause the agent to estimate lower capability probabilities incorrectly and thus miss more true positive instances. The three estimation strategies for other configurations show similar *Precision* and *Recall*.

Then, let us focus on whether different exploration methods cause performance differences by checking the gap between the two lines in each subplot, where the black line represents RLC and the red line represents RLC-Policy. The performance difference caused by different exploration methods arises primarily in two scenarios.

One is the initial stage of the learning process. The RLC-Policy is significantly better than RLC. In addition, when the exploration method is EST-Action, the RLC-policy yields obvious advantages in *Recall* compared to RLC. In other configurations or the middle and late stages of the learning process, this performance gap is not apparent due to different exploration methods. Because accurately estimating the probabilities of human capabilities requires a convergent policy, the variation in estimation results is primarily attributed to the differences in the Reinforcement Learning performance of agents employing distinct configurations. The subsequent subsections will delve deeper into the performance metrics linked with Reinforcement Learning.

Figure 5.7, Figure 5.8 and Figure 5.9 present *Accuracy* and *Hamming Loss* achieved by the proposed estimation strategies. The x-axis of all these subplots maintains its representation of the 20 learning intervals. The fluctuations on *Accuracy* and *Hamming Loss* are minimal. These three figures show that *Accuracy* experiences an upward trajectory, while *Hamming Loss* decreases. These two metrics are similar to *Precision* and *Recall*. The differences in *Accuracy* and *Hamming Loss* between different configurations are concentrated in the initial stage of the learning process. This gap is also related to the convergence of policy in Reinforcement Learning.

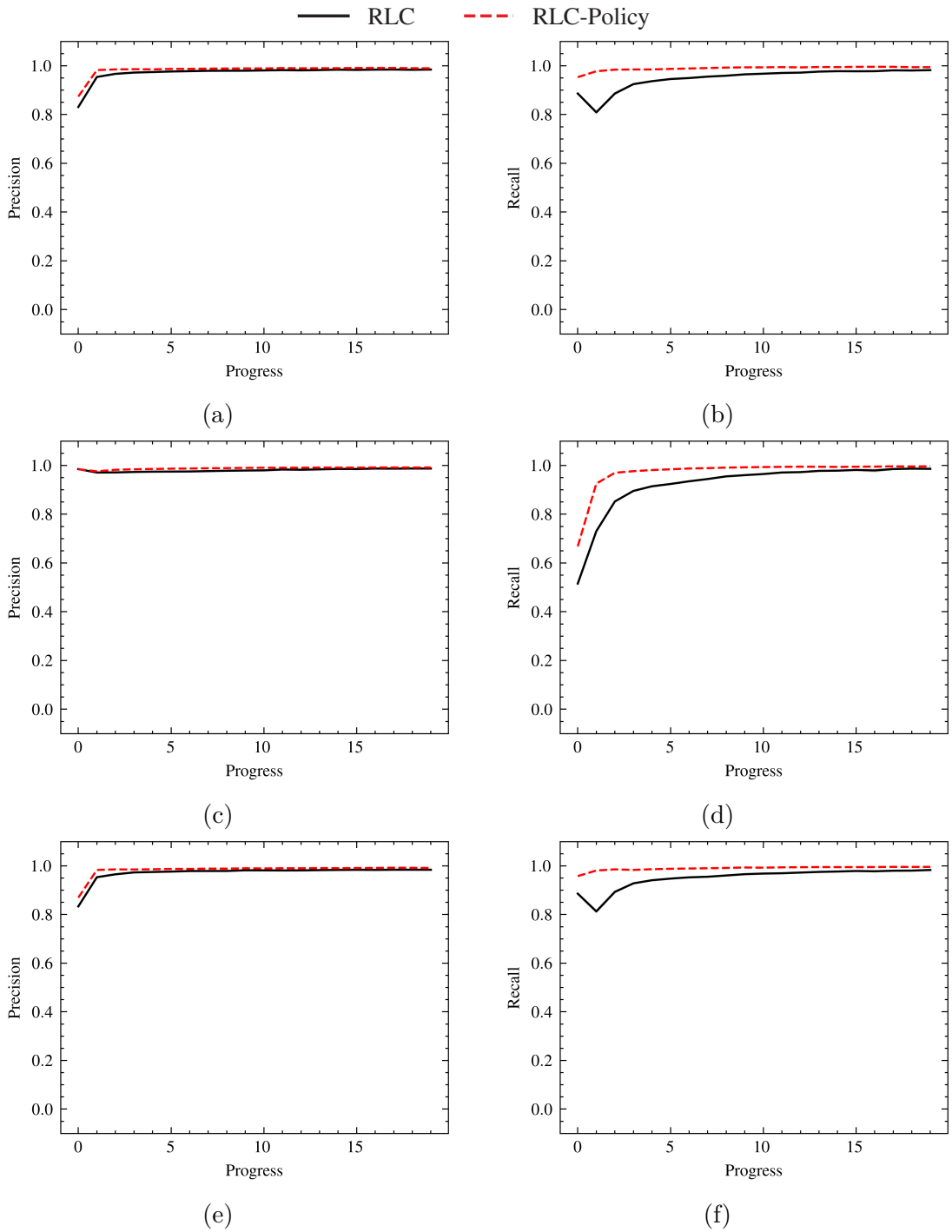


Fig. 5.4 The *Precision* and *Recall* in the navigation task (Q-learning) applied EST-Action: (a)(b) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{1, 1\})$. (c)(d) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{0, 0\})$. (e)(f) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$.

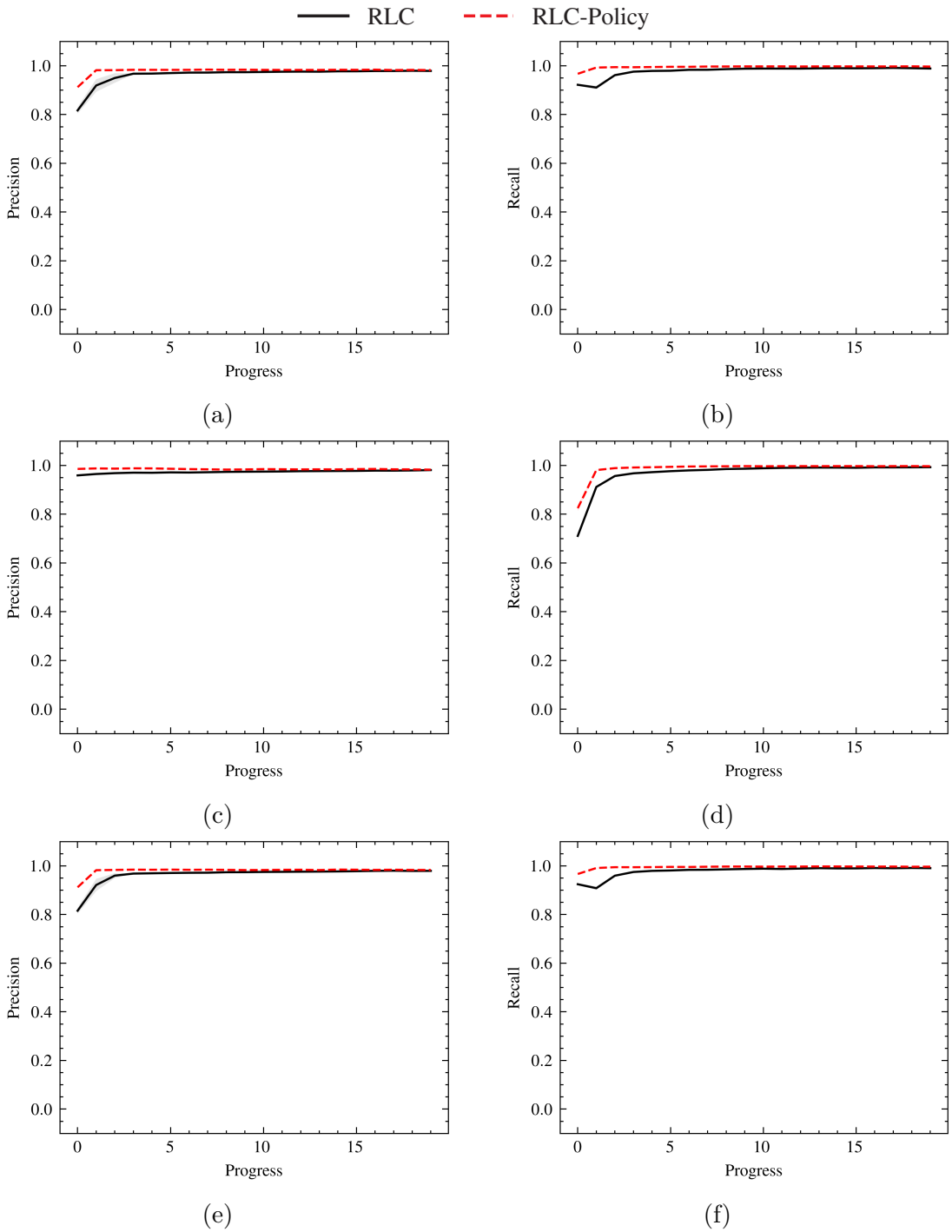


Fig. 5.5 The *Precision* and *Recall* in the navigation task (Q-learning) applied EST-State-Policy: (a)(b) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{1,1\})$. (c)(d) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{0,0\})$. (e)(f) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$.

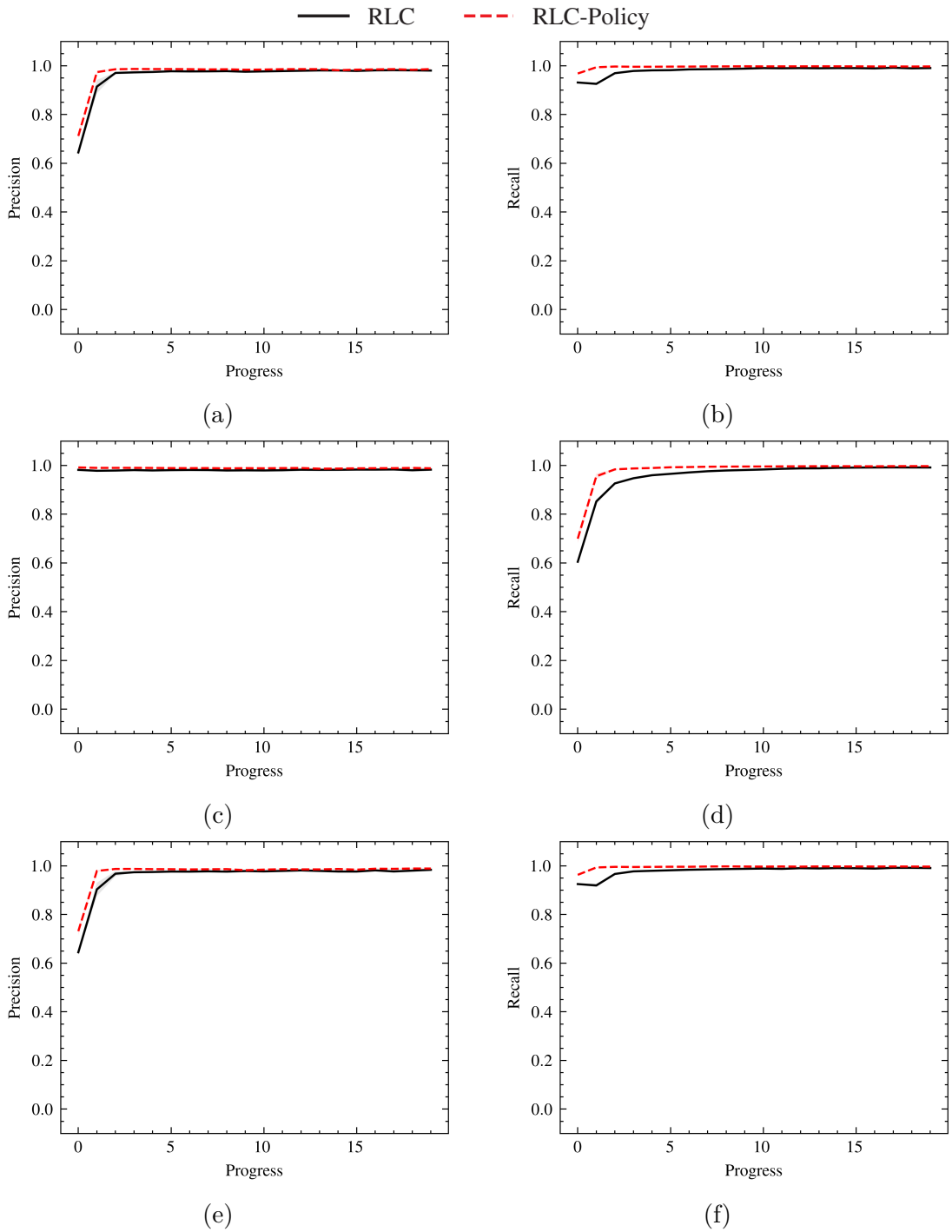


Fig. 5.6 The *Precision* and *Recall* in the navigation task (Q-learning) applied EST-Policy: (a)(b) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{1, 1\})$. (c)(d) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{0, 0\})$. (e)(f) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$.

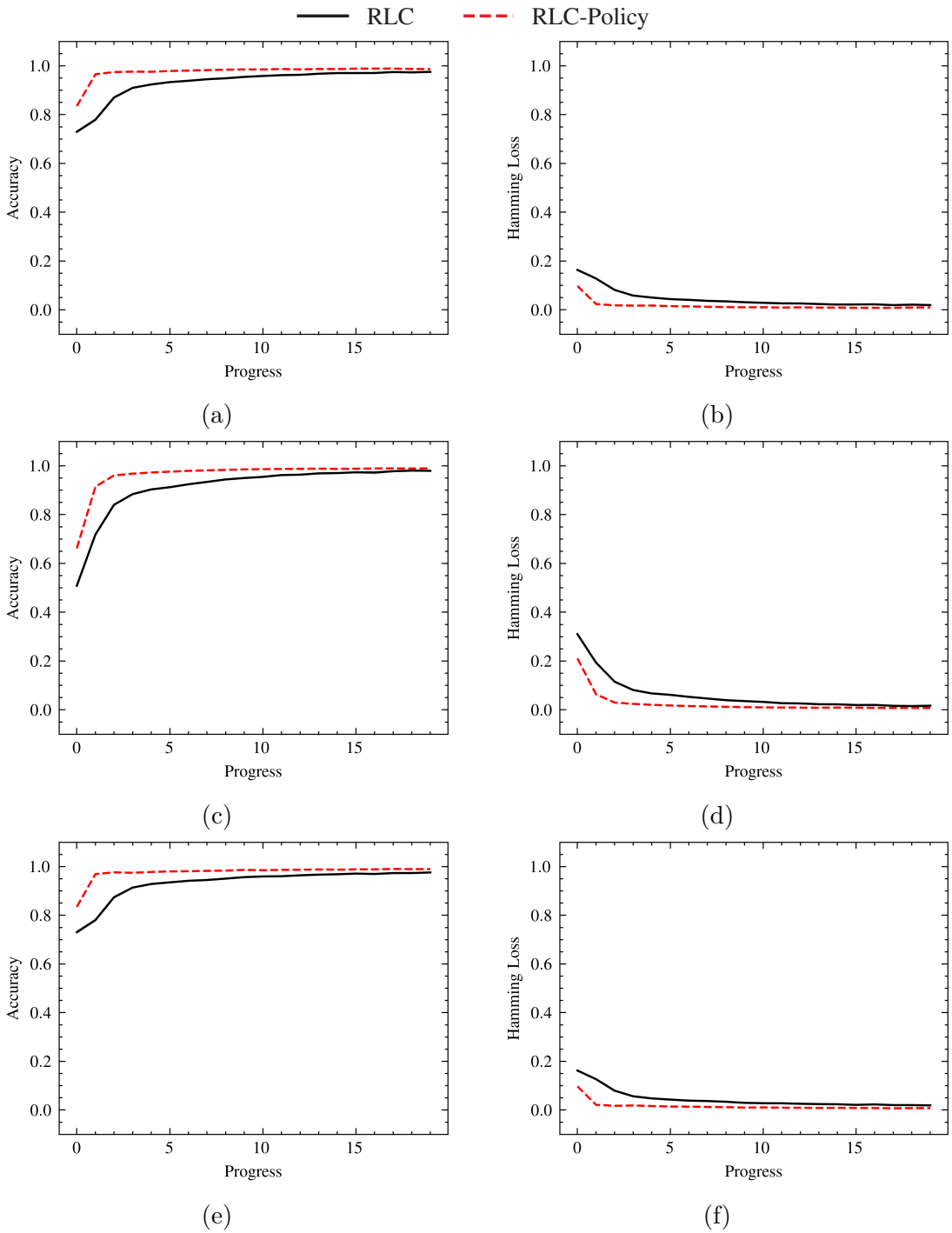


Fig. 5.7 The *Accuracy* and *Hamming Loss* in the navigation task (Q-learning) applied EST-Action: (a)(b) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{1,1\})$. (c)(d) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{0,0\})$. (e)(f) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$.

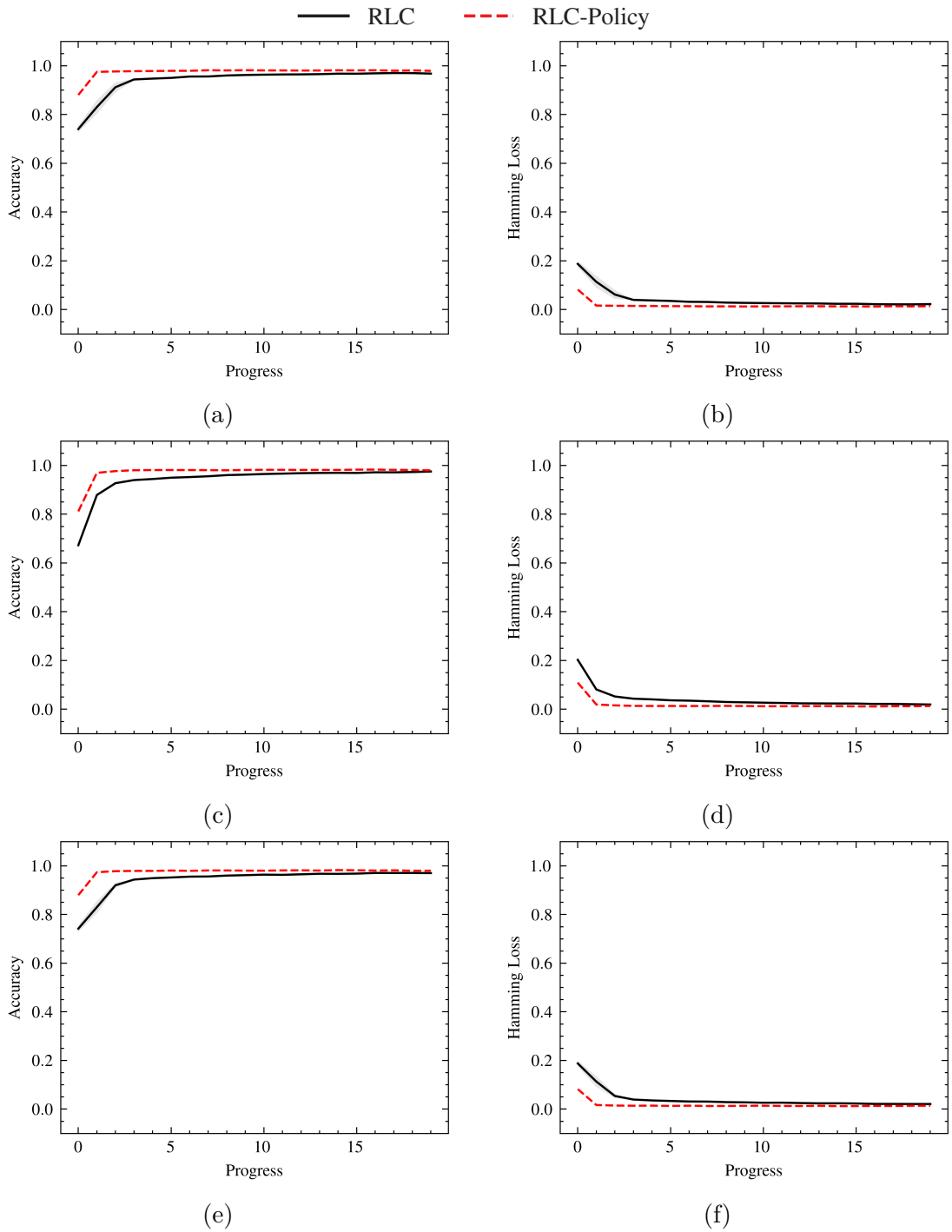


Fig. 5.8 The *Accuracy* and *Hamming Loss* in the navigation task (Q-learning) applied EST-State-Policy: (a)(b) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{1, 1\})$. (c)(d) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{0, 0\})$. (e)(f) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$.

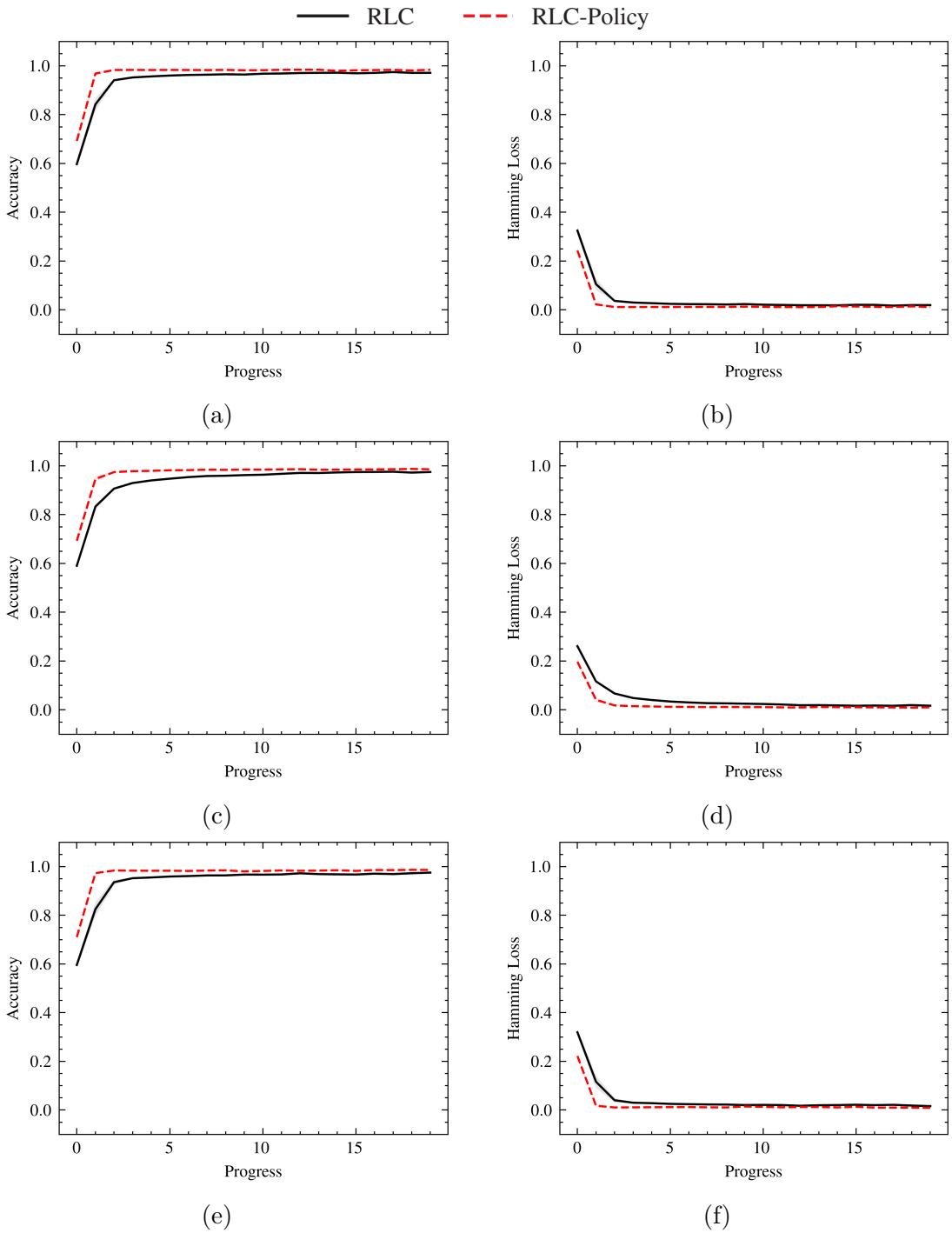


Fig. 5.9 The *Accuracy* and *Hamming Loss* in the navigation task (Q-learning) applied EST-Policy: (a)(b) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{1,1\})$. (c)(d) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{0,0\})$. (e)(f) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$.

The above evaluation metrics *Precision*, *Recall*, *Accuracy*, and *Hamming Loss* reflect that our framework can effectively estimate the probabilities of human capabilities in this navigation task. Now, let us specifically investigate how probabilities of capabilities change over time during the learning process. Figure 5.10, Figure 5.11, and Figure 5.12 show a visualization of the capability probability convergence using different estimation strategies, where the black line represents the probability of `c_fast`, and the red line represents the probability of `c_open`. All agents in these figures were initialized with the pre-trained policy $\pi^*(a \mid s, random)$ and applied RLC-Policy. The estimates in all subplots gradually converge to the correct side. Although the numbers on the x-axis are large, most probabilities of capabilities converged at an early stage. Overall, EST-State-Policy and EST-Policy showed more promising convergence rates and stability than EST-Action.

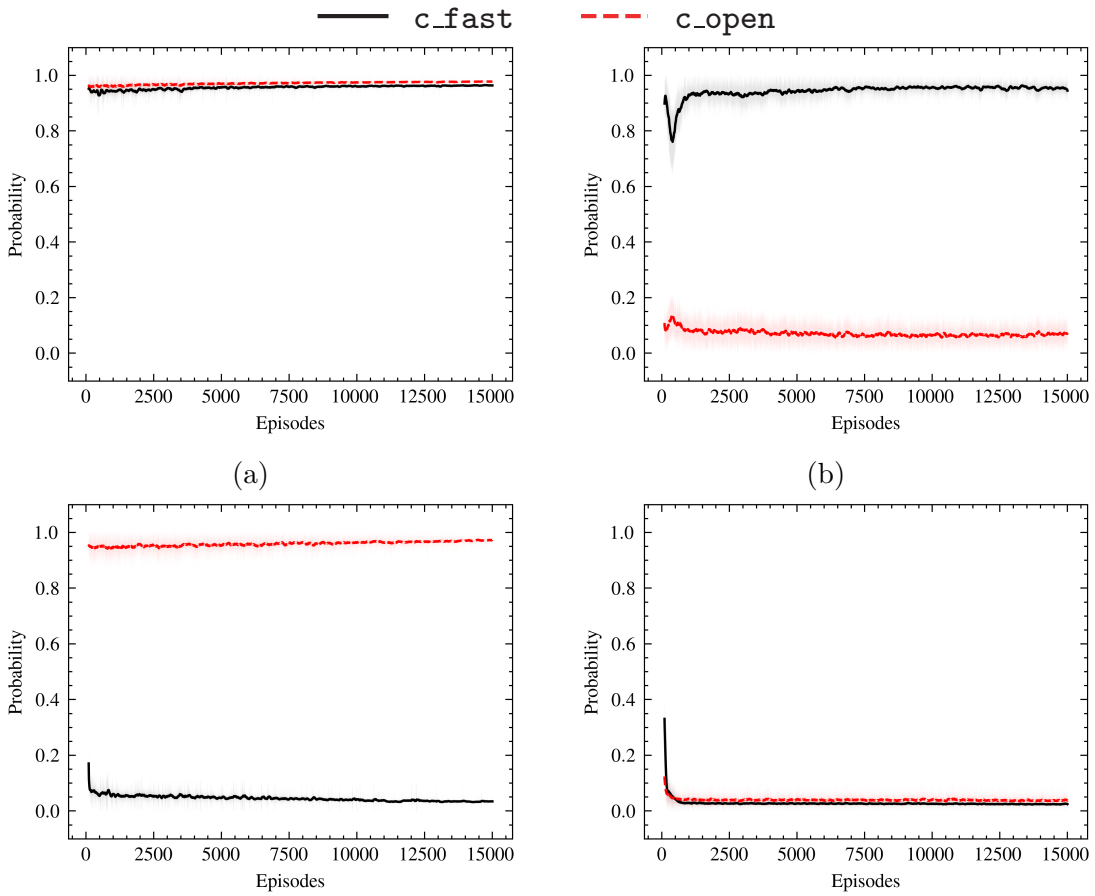


Fig. 5.10 The estimates of capabilities in the navigation task (Q-learning) applied EST-Action and RLC-Policy, where all agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$: (a) The true collaborator's capability set was $\{\texttt{c\_fast} = 1, \texttt{c\_open} = 1\}$. (b) The true collaborator's capability set was $\{\texttt{c\_fast} = 1, \texttt{c\_open} = 0\}$. (c) The true collaborator's capability set was $\{\texttt{c\_fast} = 0, \texttt{c\_open} = 1\}$. (d) The true collaborator's capability set was $\{\texttt{c\_fast} = 0, \texttt{c\_open} = 0\}$.

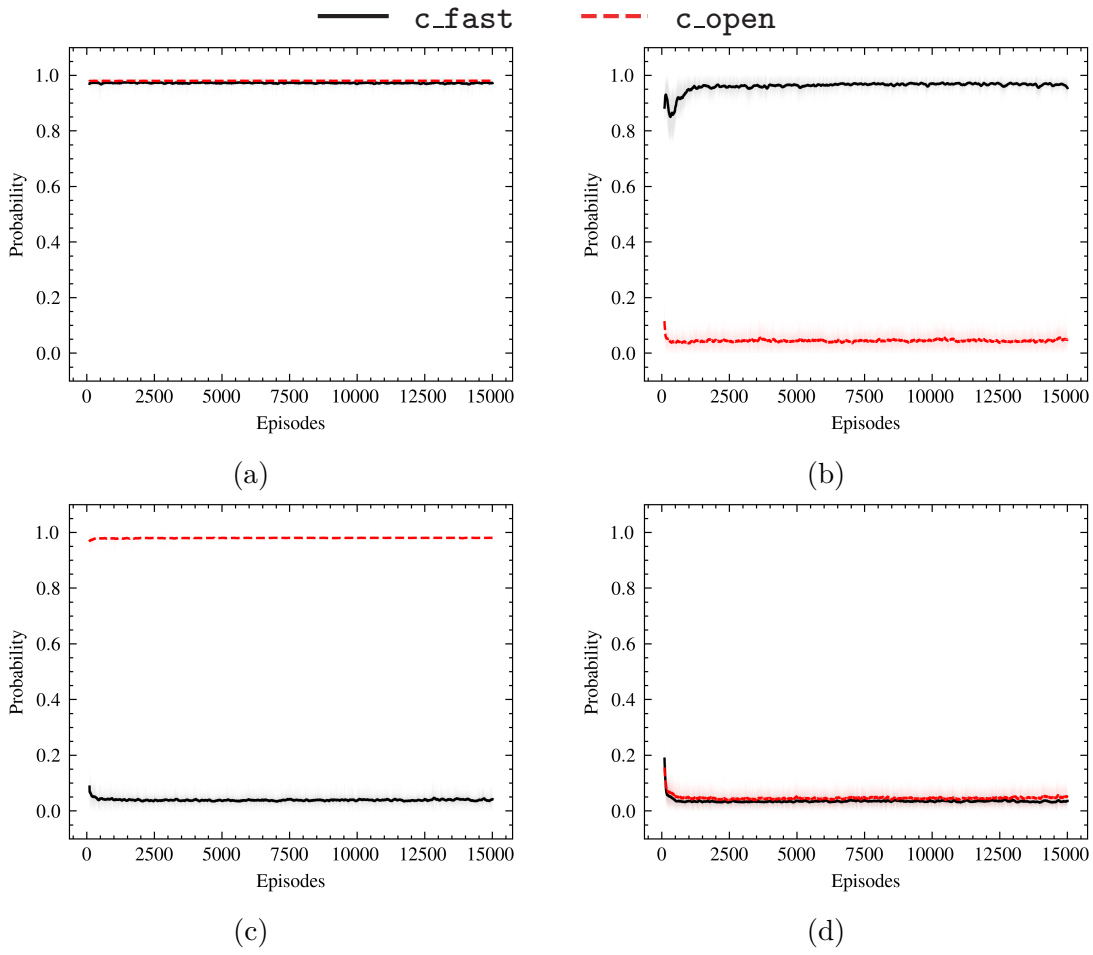


Fig. 5.11 The estimates of capabilities in the navigation task (Q-learning) applied EST-State-Policy and RLC-Policy, where all agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$: (a) The true collaborator's capability set was $\{\texttt{c\_fast} = 1, \texttt{c\_open} = 1\}$. (b) The true collaborator's capability set was $\{\texttt{c\_fast} = 1, \texttt{c\_open} = 0\}$. (c) The true collaborator's capability set was $\{\texttt{c\_fast} = 0, \texttt{c\_open} = 1\}$. (d) The true collaborator's capability set was $\{\texttt{c\_fast} = 0, \texttt{c\_open} = 0\}$.

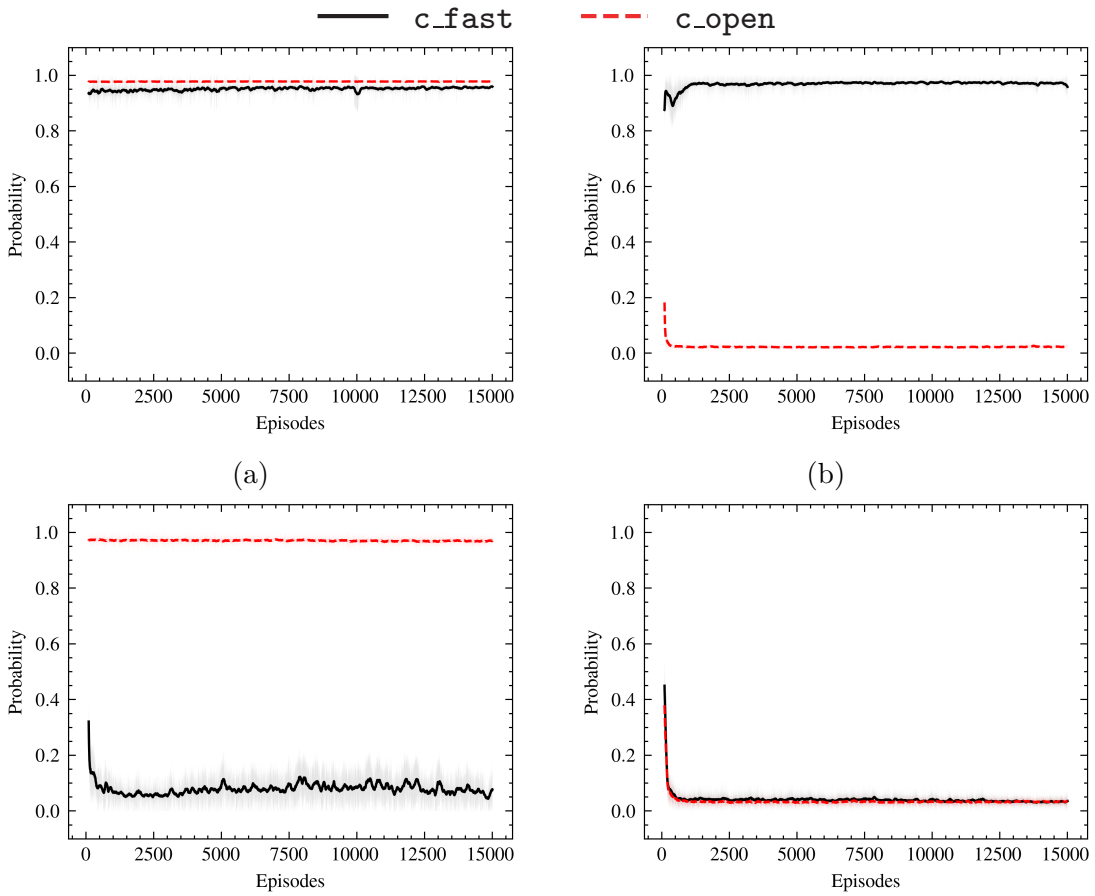


Fig. 5.12 The estimates of capabilities in the navigation task (Q-learning) applied EST-Policy and RLC-Policy, where all agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$: (a) The true collaborator's capability set was $\{\texttt{c\_fast} = 1, \texttt{c\_open} = 1\}$. (b) The true collaborator's capability set was $\{\texttt{c\_fast} = 1, \texttt{c\_open} = 0\}$. (c) The true collaborator's capability set was $\{\texttt{c\_fast} = 0, \texttt{c\_open} = 1\}$. (d) The true collaborator's capability set was $\{\texttt{c\_fast} = 0, \texttt{c\_open} = 0\}$.

We continued investigating changes in capability probabilities over time in our DQN implementation with a sensitivity analysis regarding the initial policy selection. Figure 5.13, Figure 5.14 and Figure 5.15 provide a visualization of the capability probability convergence using EST-Policy with different initial policies under DQN implementation. These three figures show the average result across configurations, accompanied by 95% confidence intervals, calculated based on 20 trials.

The agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{1, 1\})$ in the Figure 5.13, $\pi^*(a \mid s, \{0, 0\})$ in the Figure 5.14 and $\pi^*(a \mid s, random)$ in the Figure 5.15. By analyzing these three figures, we can find that at the beginning of the task, because agents had not sufficiently learned, the initial policies at this time had not been improved to the optimal policies, so the estimated results were affected by both the prior probabilities, and the bias from initial policies. This impact is evident in Figure 5.14, because when the initial policy was $\pi^*(a \mid s, \{0, 0\})$, the agent exploited cautious actions at the beginning of the task. We will continue to discuss the impact of the different initial policies on Reinforcement Learning in Section 5.2.4. Although these probabilities fluctuate in all subplots, they constantly fluctuate on the correct side or eventually converge to the correct side. It is advised not to disable actions prematurely during exploitation when dealing with unstable or unreliable capability probability estimates. This caution arises from the potential risk of incorrectly turning off certain actions. Once these actions are disabled erroneously, they tend not to be restored, given that only trajectories encompassing inappropriate states and actions are used for estimation. Therefore, when dealing with uncertain and unstable probabilities, we advise avoiding disabling actions during the exploitation process while continuing to operate capability-guided exploration. This practice does not affect the convergence of capability probabilities or improve RL performance. We will show these results when evaluating RL performance.

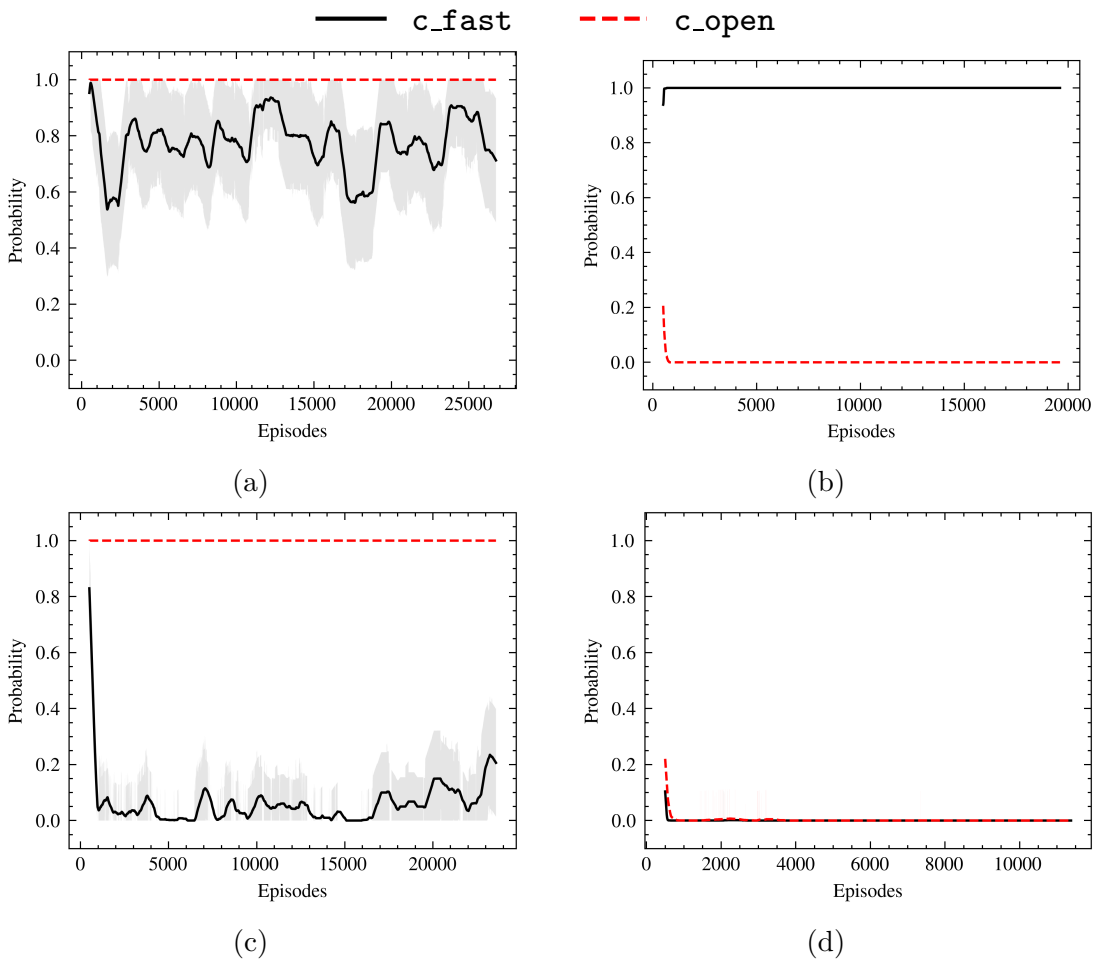


Fig. 5.13 The estimates of capabilities in the navigation task (DQN) applied EST-Policy, where all agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{1, 1\})$: (a) The true collaborator's capability set was $\{\texttt{c\_fast} = 1, \texttt{c\_open} = 1\}$. (b) The true collaborator's capability set was $\{\texttt{c\_fast} = 1, \texttt{c\_open} = 0\}$. (c) The true collaborator's capability set was $\{\texttt{c\_fast} = 0, \texttt{c\_open} = 1\}$. (d) The true collaborator's capability set was $\{\texttt{c\_fast} = 0, \texttt{c\_open} = 0\}$.

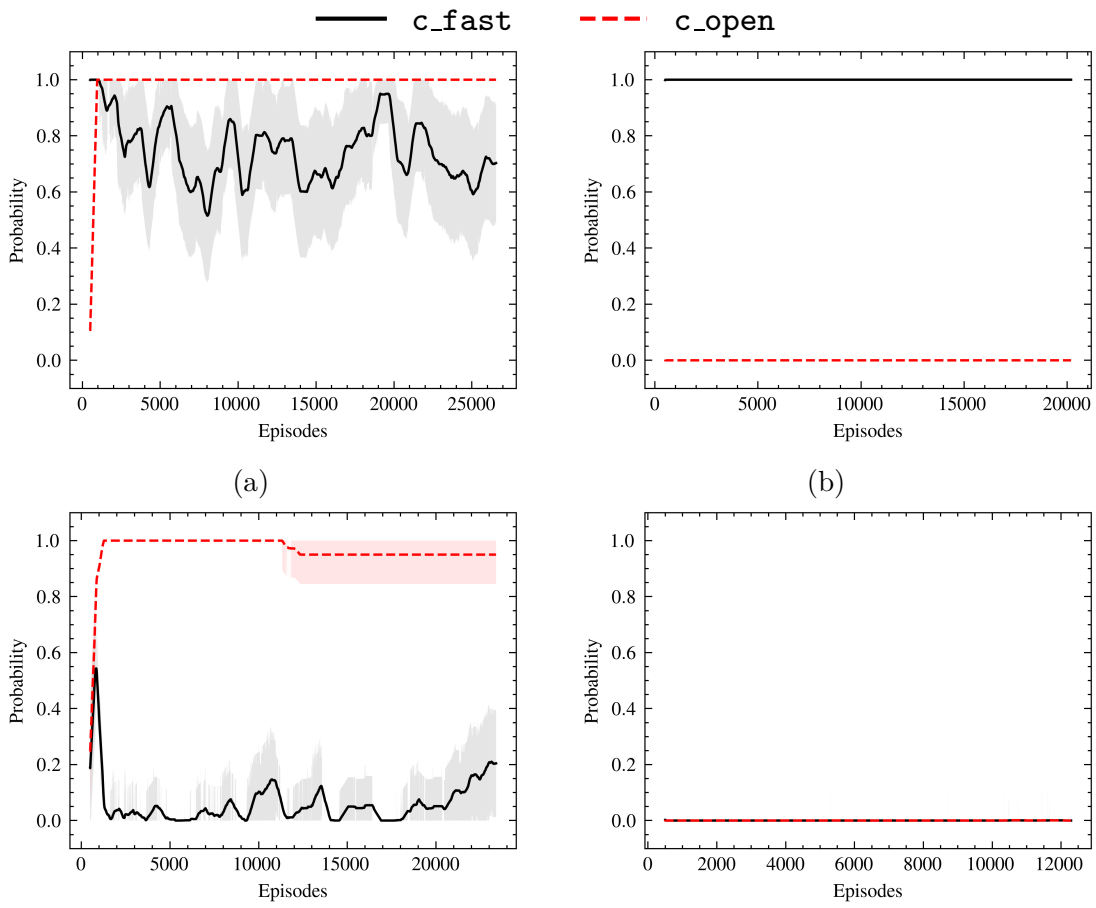


Fig. 5.14 The estimates of capabilities in the navigation task (DQN) applied EST-Policy, where all agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{0, 0\})$: (a) The true collaborator's capability set was $\{\texttt{c\_fast} = 1, \texttt{c\_open} = 1\}$. (b) The true collaborator's capability set was $\{\texttt{c\_fast} = 1, \texttt{c\_open} = 0\}$. (c) The true collaborator's capability set was $\{\texttt{c\_fast} = 0, \texttt{c\_open} = 1\}$. (d) The true collaborator's capability set was $\{\texttt{c\_fast} = 0, \texttt{c\_open} = 0\}$.

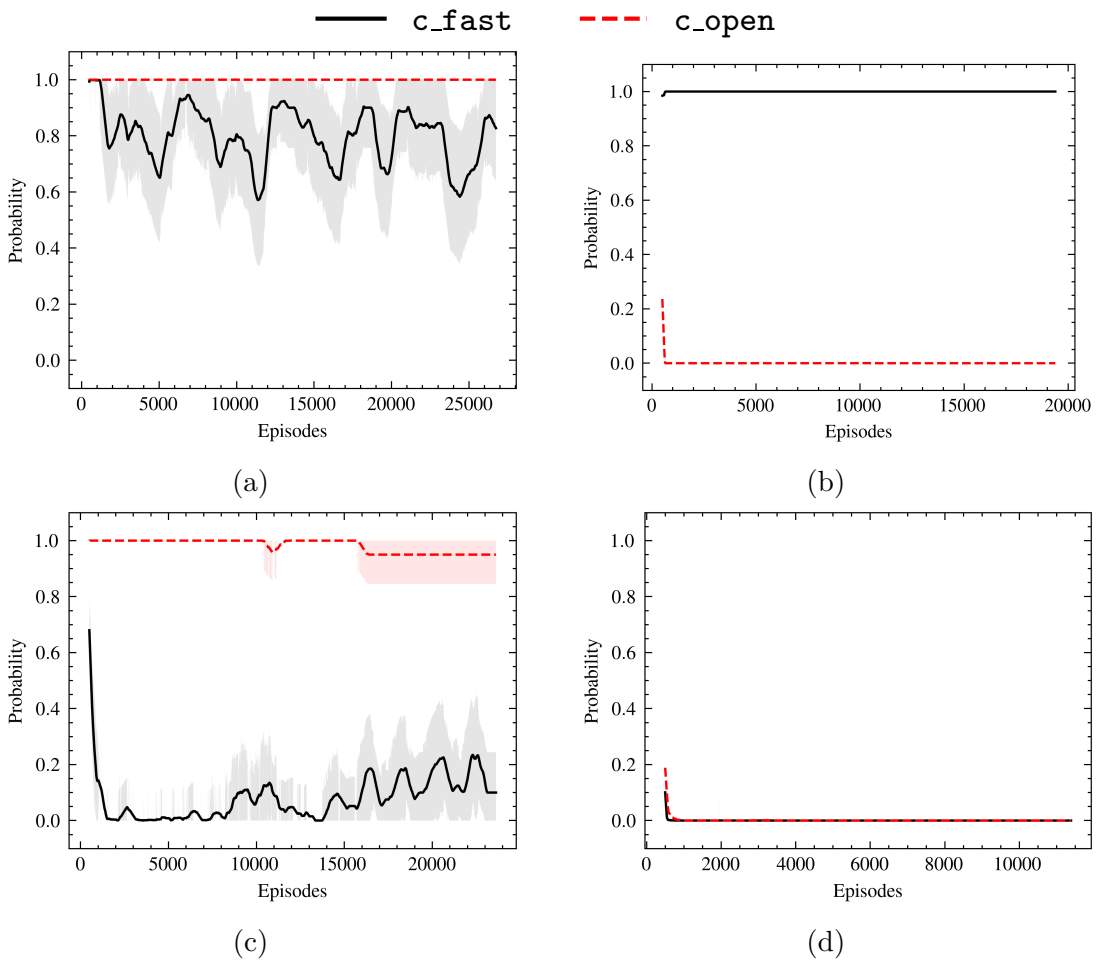


Fig. 5.15 The estimates of capabilities in the navigation task (DQN) applied EST-Policy, where all agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$: (a) The true collaborator's capability set was $\{\texttt{c\_fast} = 1, \texttt{c\_open} = 1\}$. (b) The true collaborator's capability set was $\{\texttt{c\_fast} = 1, \texttt{c\_open} = 0\}$. (c) The true collaborator's capability set was $\{\texttt{c\_fast} = 0, \texttt{c\_open} = 1\}$. (d) The true collaborator's capability set was $\{\texttt{c\_fast} = 0, \texttt{c\_open} = 0\}$.

5.2.4 RL Performance Evaluation

In this subsection, our primary objective is to run an ablation study over the different configurations of our framework and determine whether RL agents applying our framework show enhanced learning performance compared to normal RL.

We first evaluated our framework's RL performance through the Q-learning results. The Figures 5.16, 5.17 and 5.18 show the *Return* and taken steps per episode of the normal RL, RLC, and RLC-Policy. For capability estimation strategies, the agents used EST-Action in Figure 5.16, EST-State-Policy in Figure 5.17, and EST-Policy in Figure 5.18. These figures are accompanied by 95% confidence intervals, computed from 30 separate trial runs.

Across all three figures, all agents demonstrated effective learning ability. The agents did spend a certain number of episodes to learn the optimal policy, yet this number remains considerably fewer than the episodes expended during the pre-training phase. Various factors, such as hyper-parameters and reward shaping, influence the convergence rate of RL. In this work, we refrain from delving into optimizing these aspects. We focus on the contribution of capability estimation and capability-guided exploration to the convergence rate.

We shall begin by assessing the impact of different exploration methods on RL performance by comparing the difference between the three lines in each subplot, where black is for normal RL, red is for RLC, and blue is for RLC-Policy. During the early stages of learning, both RLC and RLC-Policy outperformed normal RL in terms of *Return*. At the same time, RLC-Policy consistently obtained higher *Return* and fewer steps than RLC. As learning progresses, all agents achieve similar results. However, RLC-Policy converged faster with fewer steps per episode across all cases. Capability estimation and capability-guided exploration enhance the agent's generalization ability, allowing the agent to explore reasonable actions before the action's values are refined, especially those states that the agent has not visited yet.

Next, we conduct a sensitivity analysis regarding different initial policies on RL performance by examining the difference between subplot pairs: $\{(a), (b)\}$, $\{(c), (d)\}$ and $\{(e), (f)\}$ in each figure. The convergence of online RL remains unaffected by the initial policy choice. Nevertheless, a well-chosen initial policy can facilitate learning and empower the agent to make early-stage decisions that yield higher *Return* and fewer steps. In all three figures, it is evident that agents initialized by the pre-trained policy $\pi^*(a \mid s, \{1, 1\})$ received the lowest *Return* during the early stages of learning. This outcome arose from the bias of the initial policy, that is, stubbornly performing the optimal actions `*_fast` and `open_door`. The agents did not care for the collaborator

with `c_fast` = 0, and persisted in performing `open_door`, even when the collaborator's assistance was unavailable. Thus, the agents must accumulate enough negative rewards to improve their policy. This type of policy improvement, driven by penalties imposed on the agent's actions, can still be effective even when there is insufficient exploration. On the contrary, when the initial policies were $\pi^*(a \mid s, \{0, 0\})$, the agents achieved improved results because they took cautious actions derived from this initial policy, which tends to drive at reduced speeds to avoid penalties. As a result, the agents must rely on random exploration to discover actions that lead to higher rewards. Without sufficient explorations, the agent may stick to a cautious policy and be unable to escape from the local optimum. Hence, the agents using $\pi^*(a \mid s, random)$ as the initial policy may have overall solid performances.

Upon a comprehensive comparison of these three figures, we can find that the selections in capability estimation strategies had minimal impact on the RL performance of this navigation task. Because all three estimation strategies infer correct capability estimates in this task.

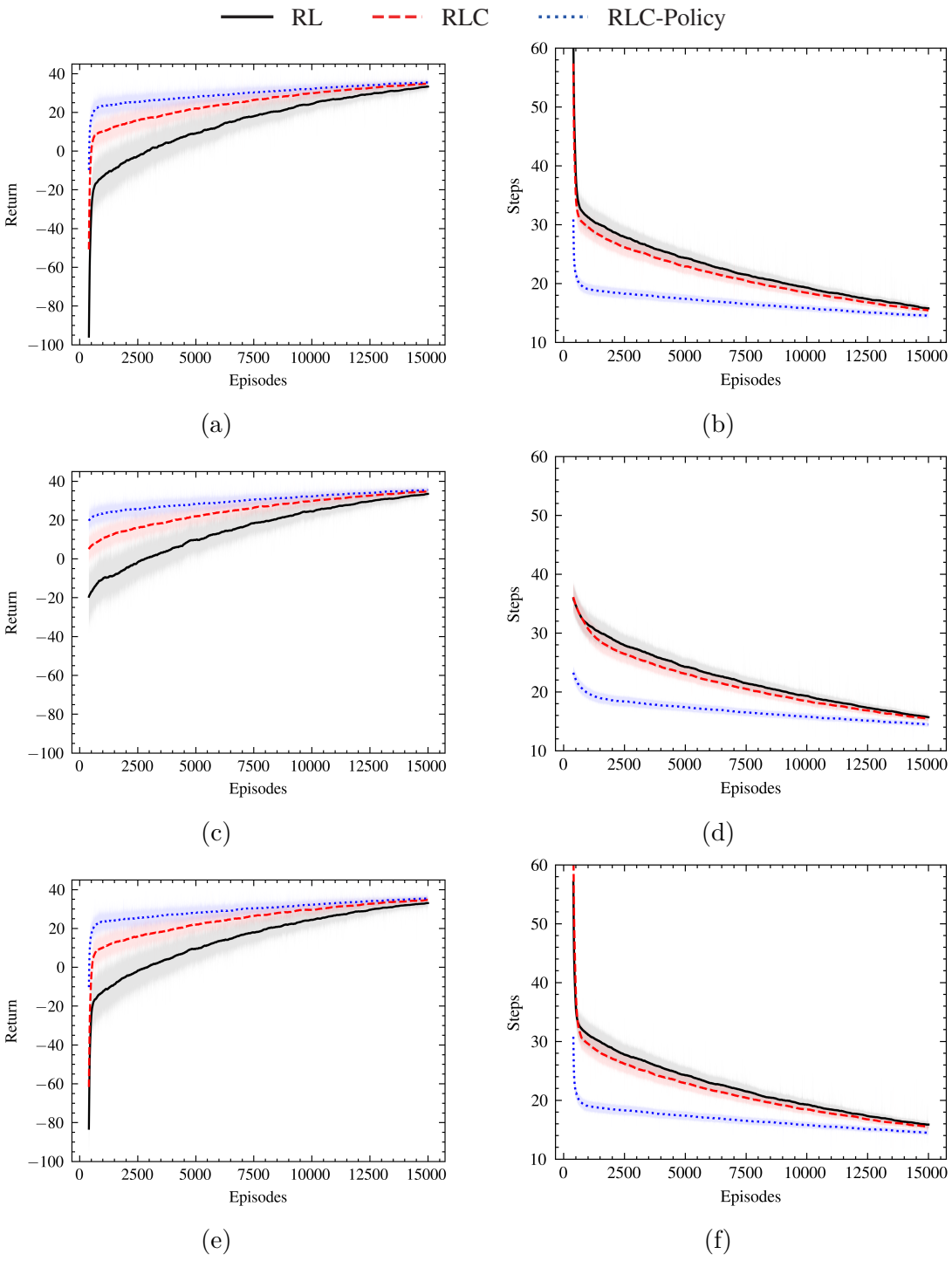


Fig. 5.16 The *Return* and the number of steps per episode in the navigation task (Q-learning), where EST-Action was applied: (a)(b) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{1, 1\})$. (c)(d) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{0, 0\})$. (e)(f) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$.

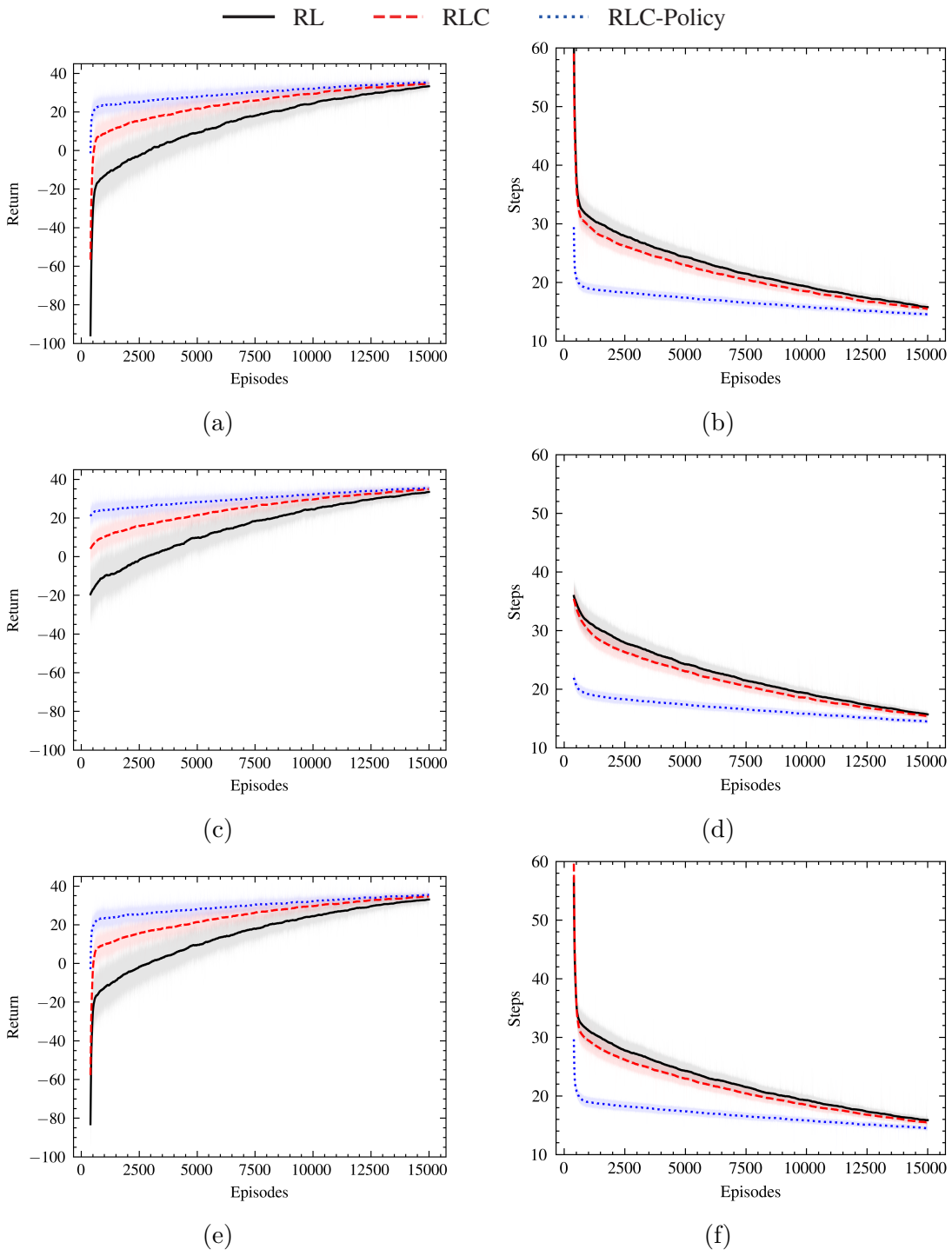


Fig. 5.17 The *Return* and the number of steps per episode in the navigation task (Q-learning), where EST-State-Policy was applied: (a)(b) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{1, 1\})$. (c)(d) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{0, 0\})$. (e)(f) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$.

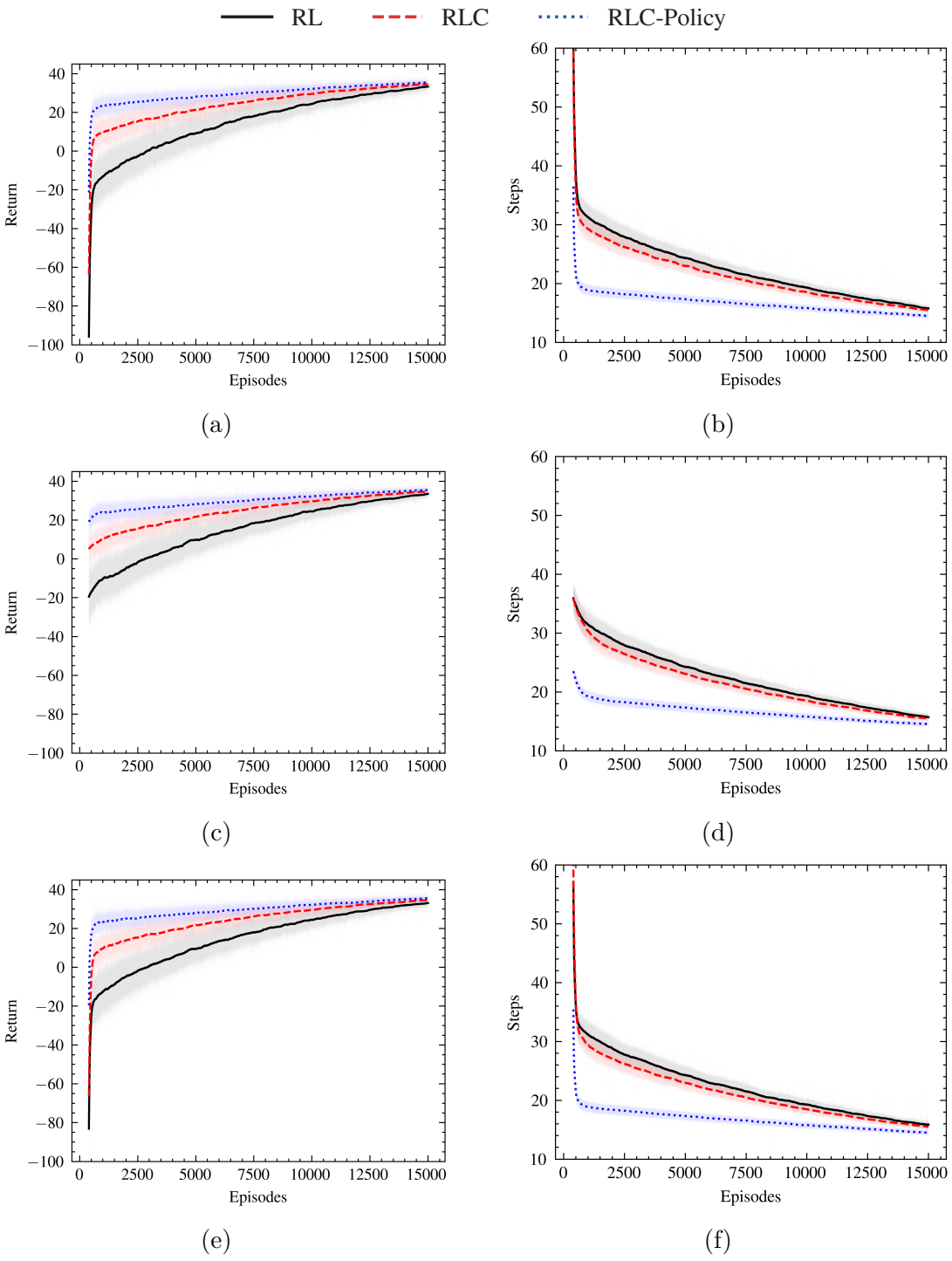


Fig. 5.18 The *Return* and the number of steps per episode in the navigation task (Q-learning), where EST-Policy was applied: (a)(b) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{1, 1\})$. (c)(d) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{0, 0\})$. (e)(f) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$.

Moving forward, let us see the results of DQN implementation. Figure 5.19 shows the *Return* and the number of steps taken per episode of the normal RL, RLC, and RLC-Policy. All subplots in this figure are accompanied by 95% confidence intervals, computed from 20 separate trial runs. By comparing the gap among normal RL, RLC, and RLC-Policy in each subplot, we can continue to conclude that RLC and RLC-Policy yield higher *Return* with fewer steps than normal RL. RLC-Policy tends to deliver even better enhancements. Moreover, when comparing the convergence rates of the DQN agents and Q-learning agents, the remarkable generalization ability of DQN facilitates quicker convergence to the optimal policy using fewer episodes over Q-learning.

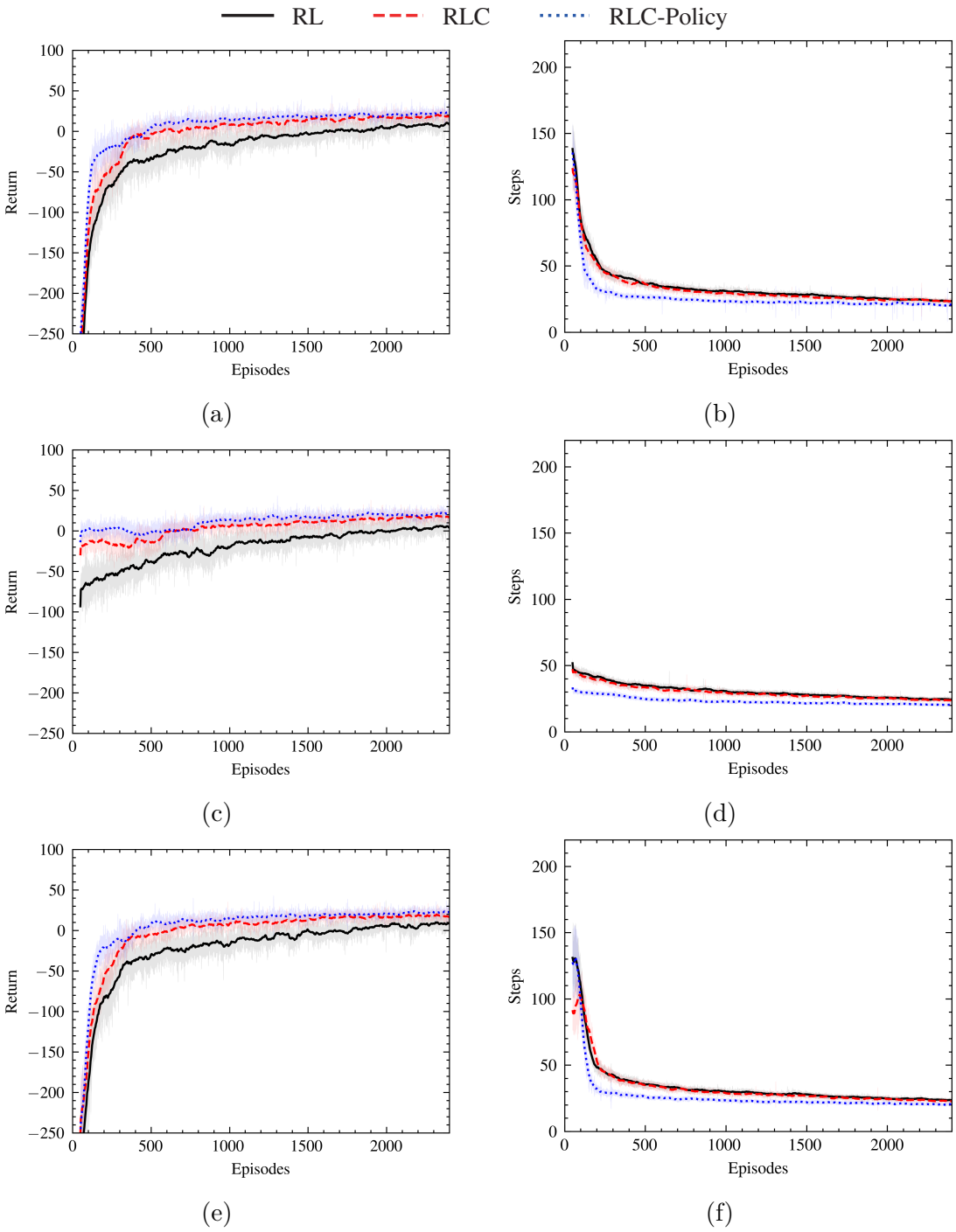


Fig. 5.19 The *Return* and the number of steps per episode in the navigation task (DQN), where EST-Policy was applied: (a)(b) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{1, 1\})$. (c)(d) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{0, 0\})$. (e)(f) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$.

5.2.5 Experiments with Lower Prior Probabilities

In Section 5.2.3 and 5.2.4, all experiments were based on high prior probabilities. To perform a sensitivity analysis on the prior probabilities, we test our framework's performance under low prior probabilities in the following experiments of this subsection. The prior probability of the combination $\{\texttt{c\_fast} = 0, \texttt{c\_open} = 0\}$ was 0.7, and for the other combinations was 0.1.

Figure 5.20 and Figure 5.21 provide a visualization of the capability probability convergence using EST-Policy with the initial policy $\pi^*(a \mid s, random)$. Each figure in this analysis contains multiple ground truths, with each row corresponding to a unique one. In each row, the left subplot visualizes the evolution of capability probabilities throughout the initial 50 episodes, while the right subplot covers the range of the 50th to the 800th episode. We can see from the above two figures that all capability probabilities start from a low prior probability we set and gradually converge to the correct side as the agent begins to learn. As discussed in the above evaluation subsections, capability probabilities are affected by both prior probabilities and initial policy, which is confirmed again in these two figures. We can observe that except for subplots (a) and (b) in Figure 5.20, when the ground truth of a certain capability probability is False, the capability probability did not directly converge to near 0. It initially exhibited a surge along the Y-axis. This unexpected fluctuation is attributed to the bias introduced by the initial policy $\pi^*(a \mid s, random)$. Since capability probabilities are unreliable at the initial stage, we did not disable actions in the exploitation process given this low prior probability configuration. And this practice has no impact on the convergence of capability probabilities.

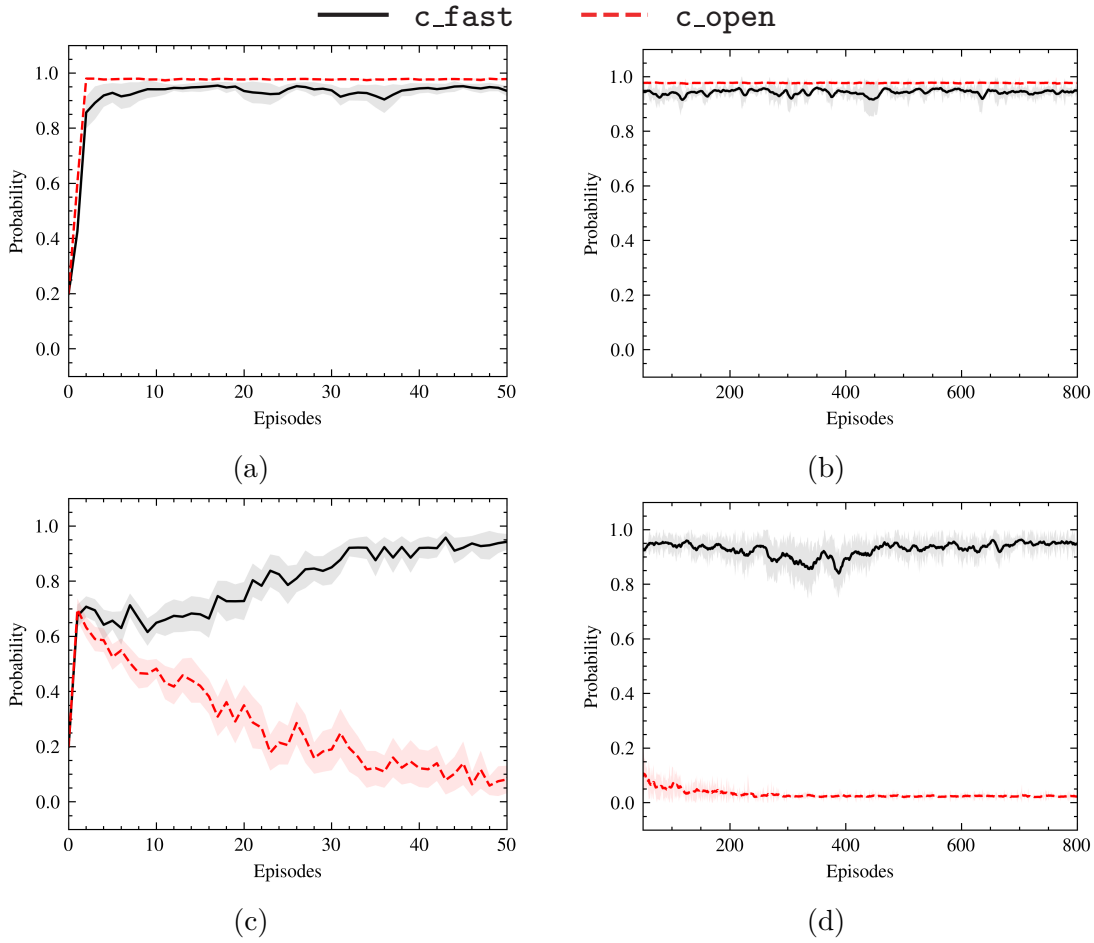


Fig. 5.20 The estimates of capabilities in the navigation task (Q-learning) applied EST-Policy and RLC-Policy, where all agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$ and lower prior probabilities: (a)(b) The true collaborator's capability set was $\{\texttt{c\_fast} = 1, \texttt{c\_open} = 1\}$. (c)(d) The true collaborator's capability set was $\{\texttt{c\_fast} = 1, \texttt{c\_open} = 0\}$.

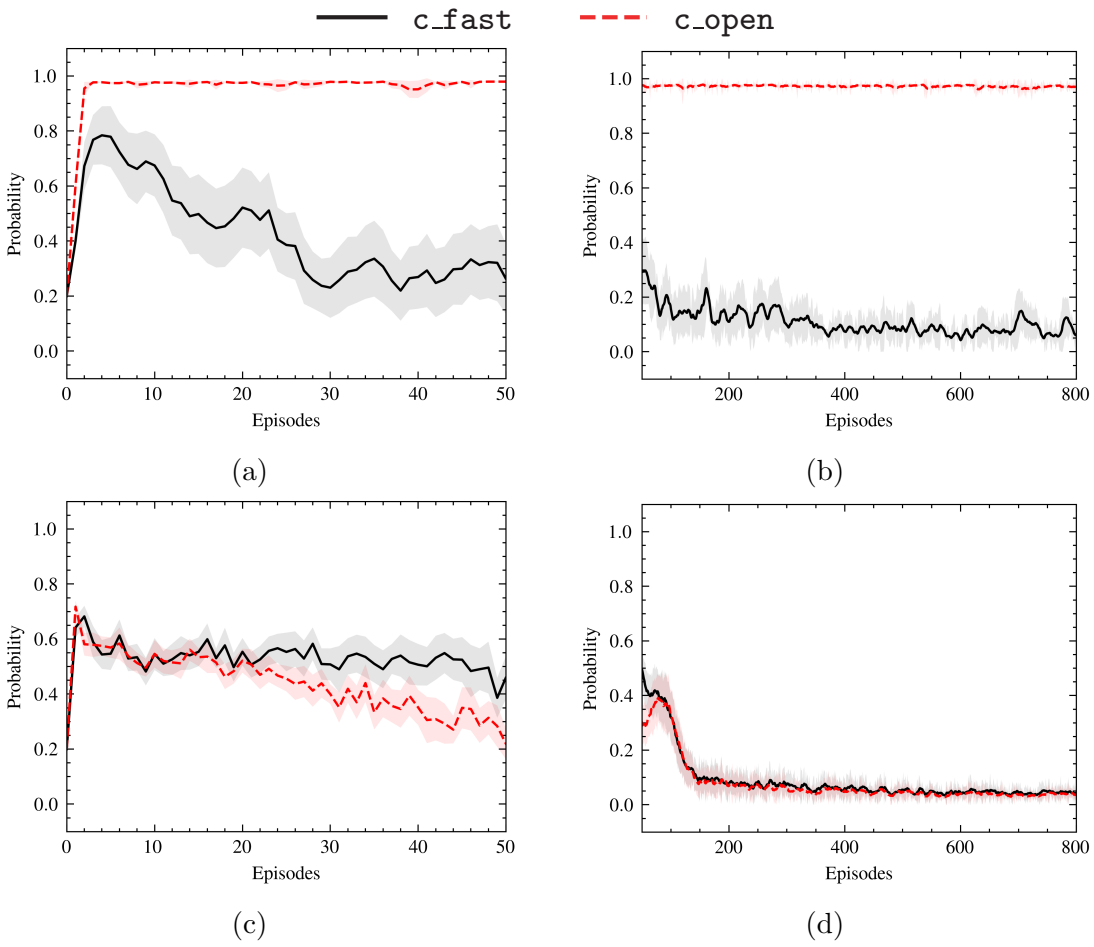


Fig. 5.21 The estimates of capabilities in the navigation task (Q-learning) applied EST-Policy and RLC-Policy, where all agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$ and lower prior probabilities: (a)(b) The true collaborator's capability set was $\{\texttt{c\_fast} = 0, \texttt{c\_open} = 1\}$.(c)(d) The true collaborator's capability set was $\{\texttt{c\_fast} = 0, \texttt{c\_open} = 0\}$.

In addition to the convergence of capability probabilities, we are also concerned with the performance of RL under conditions of low prior probabilities. Figure 5.22 shows the average return and the number of steps obtained by the agent using different initial policies when applying the EST-Policy estimation strategy. Results highlight that the practice of low prior probability does not significantly affect the performance improvement of RLC and RLC-Policy compared to the normal RL. The overall performance of RLC-Policy is still better than that of RLC.

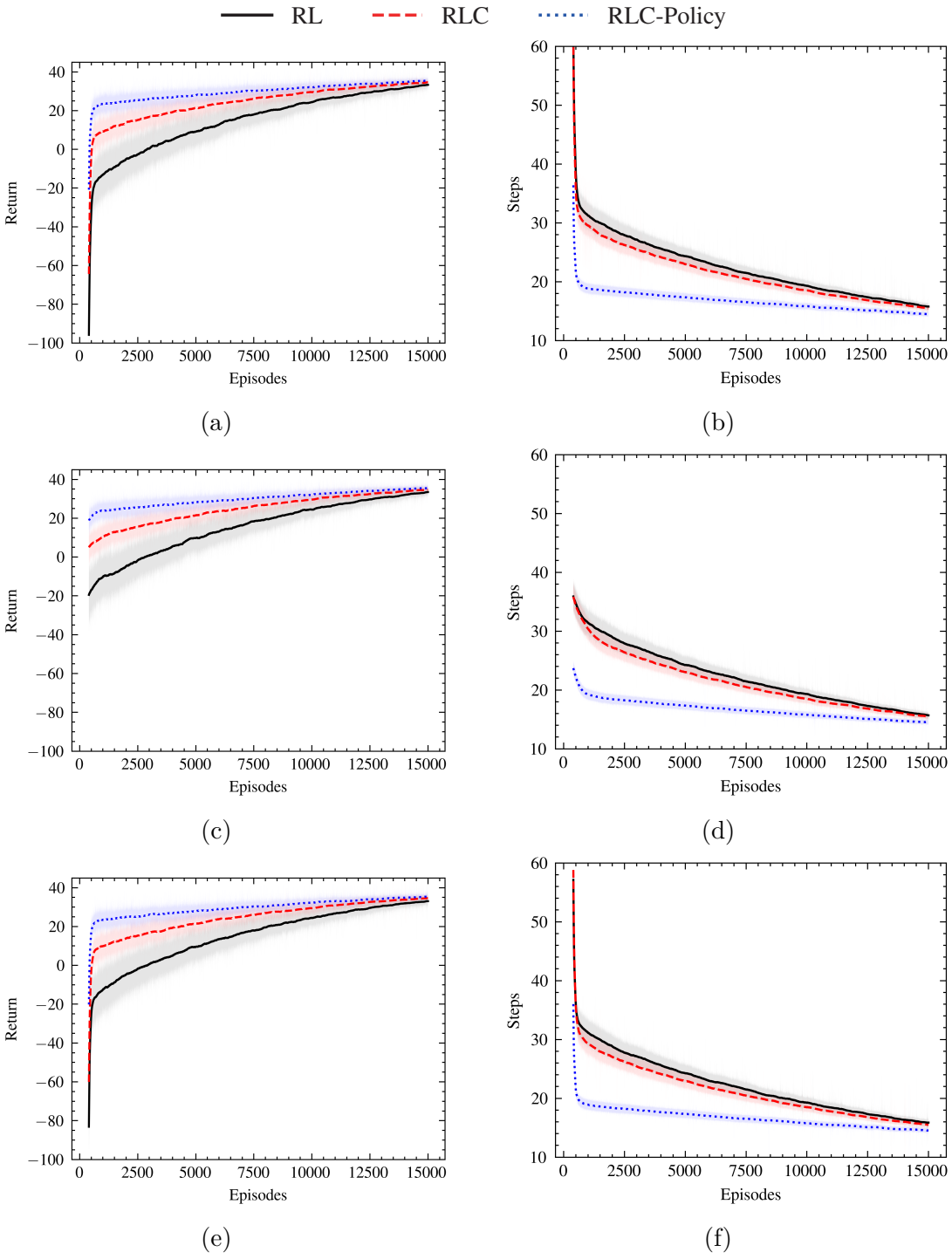


Fig. 5.22 The *Return* and the number of steps per episode in the navigation task (Q-learning), where EST-Policy and lower prior probabilities were applied: (a)(b) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{1, 1\})$. (c)(d) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{0, 0\})$. (e)(f) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$.

5.2.6 Summary

In this section, we continued introducing the robot navigation task used in the previous chapter and added more details of the experimental environment. We introduced the parameters of Q-learning and DQN implementations, followed by the experimental results. Furthermore, we conducted ablation studies across two main dimensions: estimation performance and RL performance. It allowed us to discuss the performance gaps observed among various configurations. Furthermore, we conducted a sensitivity analysis on prior probabilities and the initial policy.

5.3 Robot Manipulation Task

5.3.1 Introduction

The second experiment is a simulated collaborative manipulation task, where the collaborator and robot work together to arrange objects. The task is set in a Gazebo world, illustrated in Figure 5.23. The environment consists of a table with a 2×3 chessboard, providing six slots for placing objects. Six objects will be sorted: two red, two orange, and two blue objects.

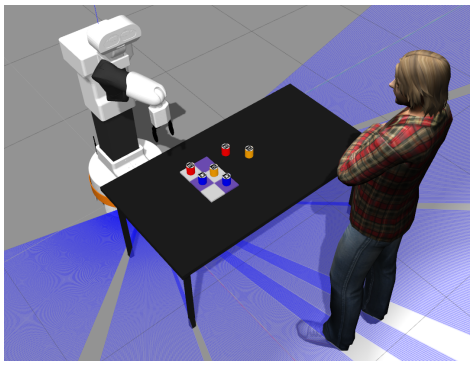


Fig. 5.23 The Gazebo world for the manipulation task.

At the beginning of the task, some objects are randomly placed on the board, ensuring a maximum of 4 slots remain vacant. The objects not placed on the board initially are set on the table. Subsequently, the robot and the collaborator take turns placing or swapping these objects' positions. The goal is to organize all the objects on the board, ensuring that objects of the same color are grouped together. The task ends when all objects are successfully grouped based on their colors. As illustrated in Figure 5.24, multiple solutions are acceptable to end the task.

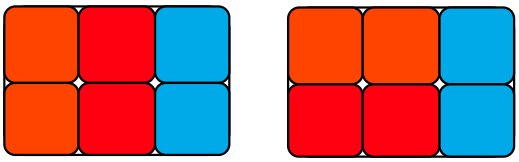


Fig. 5.24 Examples of acceptable solutions in the manipulation task.

5.3.2 MDP Settings

The state space consists of observations for six slots (2×3) on the chessboard. For the DQN implementation, we represent the colors of the slots using One-Hot encoding; otherwise, we use label encoding.

The action space contains nine actions, which can be categorized as follows:

(i) `place_{red,orange,blue}`: These actions represent that the robot places an unorganized item of the given color onto the chessboard from the table.

(ii) `swap_{red,orange,blue}`: These actions represent that the robot chooses an item on the chessboard of the given color and swaps it with an item of a different color present on the chessboard.

(iii) `ask_swap_{red,orange,blue}`: These actions represent that the robot asks the collaborator to swap the item of the given color on the chessboard.

The RL agent serves as the high-level task planner and operates in an action space that only indicates the action included in the action space and the target color. A separate motion solver handles complete information about the colors and positions of all items, whether on the chessboard or the table. It transforms the RL agent's high-level action into a sequence of optimal manipulations for the robot. By setting this motion solver, the RL agent can focus on high-level instructions without being concerned about the details of the manipulation process.

We define L_e as the number of objects with position discrepancies between the current layout and the nearest acceptable solution. The reward function for the task is designed as follows:

(i) `place_*`: When the robot performs a placing action, it receives a cost of -1. This cost discourages unnecessary actions and encourages the robot to be more strategic.

(ii) `ask_swap_*`: If the robot chooses to ask the collaborator to swap items and the collaborator responds to the robot's request, the robot receives a cost of

-0.1. This cost encourages the robot to maximize the times it relies on human assistance when available, stimulating human-robot collaborations.

(iii) `swap_*`: When the robot autonomously performs a swap, it receives a higher cost of -2. Because our TIAGo has only one robotic manipulator, the robot autonomously swapping the positions of two items requires three steps and occupies an extra storage space on the table. This penalty is to prevent the robot from engaging in inefficient swaps.

(iv) **Penalty for No Layout Variation:** If the robot's strategy does not lead to any layout variations ($\Delta L_e = 0$), it receives an additional penalty of -10. This penalty discourages the robot from getting stuck in repetitive or unproductive actions.

(v) **Penalty for Human Inactivity:** If the robot's strategy leaves the collaborator with nothing to do, the robot receives an additional penalty -5, because the robot must not aim to reduce human participation in this task.

(vi) **Goal Layout Reward:** If the robot's strategy results in the correct goal layout, it receives an additional reward of 50.

The manipulation experiment involves a more complex environment than the navigation task. Each slot can hold objects of 3 different colors and be in an empty state. Unlike the navigation task, where the starting positions of the robot and collaborator are consistent, the manipulation task randomizes its initial layout. Furthermore, the manipulation task has more terminal states. These differences not only contribute to a more dynamic and stochastic environment but also pose challenges for the RL agent in generalizing an optimal policy. Achieving effective generalization across the vast state space and adapting to the dynamic environment requires the agent to handle a broader range of scenarios and decision-making situations. As a result, the DQN implementation, rather than Q-learning, is better suited for this manipulation task.

5.3.3 Motion planner

The motion planner consistently generates a greedy plan to reduce the L_e, guided by the high-level command from the RL agent. When multiple plans are available, the planner randomly selects one for execution. This planner is incorporated not just for the robot but also for the human model. We employed Redis[3], an open-source

[3]https://redis.io/

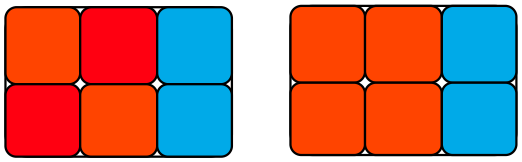
distributed in-memory key-value database, to optimize planning speed. The program stores every plan in Redis after generating it. This strategy proves highly beneficial in multi-process or multi-computing node environments as the cached plan in Redis can be accessed and shared among all planners, resulting in significant computational time savings.

5.3.4 Human Capabilities

We consider two human capabilities in this experiment: `c_green` and `c_swap`. The absence of `c_green` indicates that the collaborator cannot differentiate between red and orange colors, a condition known as Deuteranomaly [30]. The `c_swap` capability implies the ability to swap two objects in a single move, which can be affected by mobility impairment or having only one arm.

5.3.5 Human Model

The human model serves as a simulator to interact with the robot. Human capabilities influence the behaviors of the human model. If the capability `c_green` is absent, the human model will not distinguish between red and orange items, as an example shown in Figure 5.25. Similarly, the human model will not perform any swaps if the capability `c_swap` is absent. When the robot requests the human to swap two items, the accuracy of the exchange depends on the accurate perception of colors, hence influenced by the capability `c_green`.



(a) (b)

Fig. 5.25 An example of visual differences caused by `c_green`: (a) The layout seen by the collaborator `c_green` = 1. (b) The layout seen by the collaborator `c_green` = 0.

To consider that the person can make mistakes, the human model improves the correct grouping of items, reducing L_e (e.g., placing a blue item near other blue items) with a probability of 0.9, which can be mathematically represented as $\Delta L_e < 0$. However, there is also a probability of 0.1 that the human model may worsen the configuration with its actions, mathematically indicated by $\Delta L_e \geq 0$.

5.3.6 Experimental Setup

Q-learning Implementation

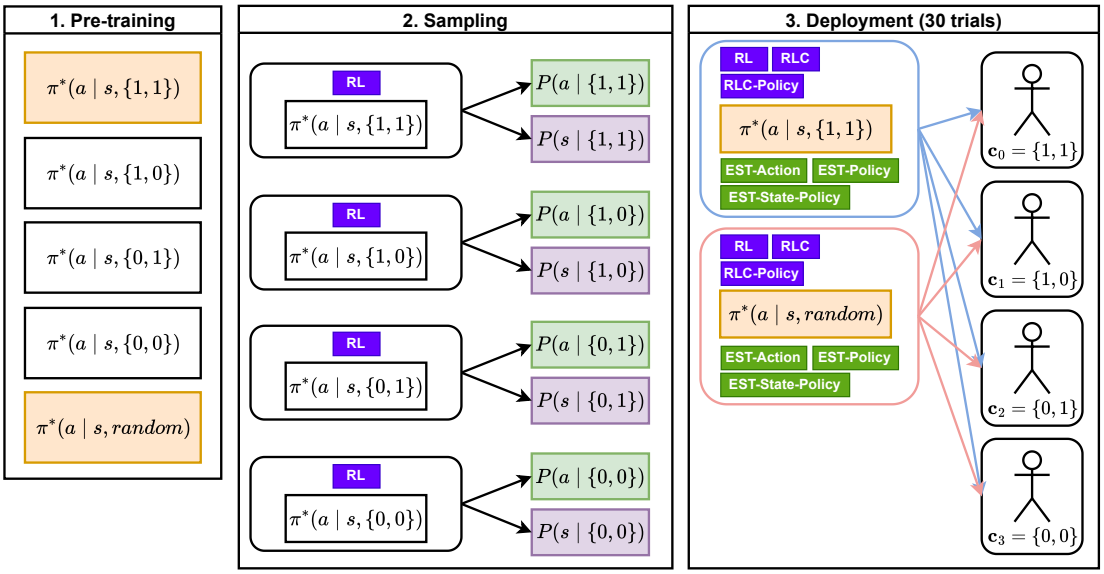


Fig. 5.26 The experimental setup of the manipulation task (Q-learning).

The experiment started with pre-training, resulting in $|2^{\mathbf{c}}| = 4$ pre-trained policies and one extra pre-trained policy for random capabilities. The pre-trained policies $\pi^*(a \mid s, \{1,1\})$ and $\pi^*(a \mid s, random)$ serve as the initial policies. For each initial policy, the RL agents were deployed separately with collaborators of all $|2^{\mathbf{c}}| = 4$ capability combinations. We tested all combinations of {EST-Action, EST-State-Policy, EST-Policy} $\times$ {RL(normal), RLC, RLC-Policy} separately in the deployment. Consequently, we had $2 \times 4 \times 3 \times 3 = 72$ deployment configurations. Figure 5.26 shows the overview of the experimental setup.

In pre-training, the agents were trained over 500,000 episodes for each capability combination. The learning rate $\alpha = 0.1$, the discount factor $\gamma = 0.7$. The maximum number of steps in a single episode was 10. The prior probability of combination $\{\texttt{c\_green} = 1, \texttt{c\_swap} = 1\}$ was 0.8, and for the other combinations was 0.06.

In sampling, the agents ran 300,000 episodes for each capability combination. The Softmax temperature $\tau = 0.1$. The smoothing factor of Laplace smoothing was 1.

In the deployment, the agents ran 80,000 episodes for each configuration. The ϵ of ϵ-greedy decayed linearly from 0.5 to 0.1. The learning rate $\alpha = 0.1$, the discount factor $\gamma = 0.7$. The maximum number of steps in a single episode was 10. The $\kappa = 0.5$ and

$\kappa_{pre} = 0.5$. The capability thresholds $\mathbf{d} = \{0.5\}^2$. The Softmax temperature $\tau = 0.1$. The batch size of estimating trajectory $l = 20$.

DQN Implementation

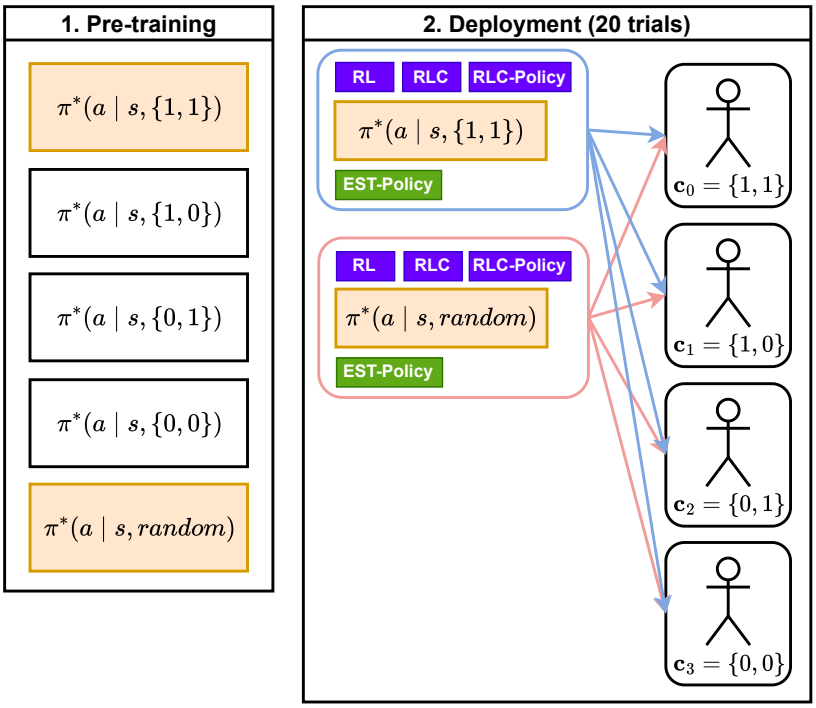


Fig. 5.27 The experimental setup of the manipulation task (DQN).

The experiment started with pre-training, resulting in $|2^{\mathbf{c}}| = 4$ pre-trained policies and one extra pre-trained policy for random capabilities. The pre-trained policies $\pi^*(a \mid s, \{1,1\})$ and $\pi^*(a \mid s, random)$ serve as the initial policies. For each initial policy, the RL agents were deployed separately with collaborators of all $|2^{\mathbf{c}}| = 4$ capability combinations. We tested all combinations of {EST-Policy} $\times$ {RL(normal), RLC, RLC-Policy} separately in the deployment. Consequently, we had $2 \times 4 \times 1 \times 3 = 24$ deployment configurations. Figure 5.27 shows the overview of the experimental setup.

In the pre-training, the agents were trained over 2,000,000 steps for each capability combination. The learning rate $\alpha = 0.001$, the discount factor $\gamma = 0.7$. The maximum number of steps in a single episode was 100. The prior probability of combination $\{\texttt{c\_fast} = 1, \texttt{c\_open} = 1\}$ was 0.8, and for the other combinations was 0.06. The batch size of learning is 2048. The ϵ initially started at 0.8 and began to decay. Once the training progress reached 80%, the epsilon was reduced to 0 and remained fixed. The policy network consisted of two layers, each with a size of 1024.

In the deployment, the agents ran 200,000 steps for each configuration. The $\kappa = 0.5$ and $\kappa_{pre} = 0.5$. The capability thresholds $\mathbf{d} = \{0.5\}^2$. The Softmax temperature $\tau = 0.01$. The batch size of trajectory $l = 4$. The ϵ initially started at 0.8 and began to decay. Once the training progress reached 80%, the epsilon was reduced to 0 and remained fixed.

5.3.7 Estimation Performance Evaluation

In this subsection, our primary focus centers on assessing *Accuracy*, *Precision* , *Recall* and *Hamming Loss*. The figures of Q-learning implementation displayed in this subsection show the average result across configurations in the Q-learning deployment, accompanied by 95% confidence intervals, calculated based on 30 trials.

Figure 5.28, Figure 5.29 and Figure 5.30 show *Precision* and *Recall* of the three estimation strategies: EST-Action, EST-State-Policy and EST-Policy. The x-axis across all these figures represents the progression of learning, segmented into 20 intervals. Specifically, $x = 0$ denotes the initial 5% of the learning process, while $x = 19$ corresponds to 100% completion.

The fluctuations of all results displayed in these figures are tiny. We first focus on the performance gaps between different estimation strategies, examining the differences among Figure 5.28, Figure 5.29 and Figure 5.30. Overall, both *Precision* and *Recall* are great as their values remain high. Nevertheless, these two metrics of EST-State-Policy change more smoothly in the initial stage than EST-Action and EST-Policy.

Subsequently, we shift our attention to whether different exploration approaches result in performance differences by reviewing the gap between the two lines in each subplot, where the black line represents RLC and the red line represents RLC-Policy. It is evident that there was no considerable difference in *Precision* and *Recall* between RLC and RLC-Policy.

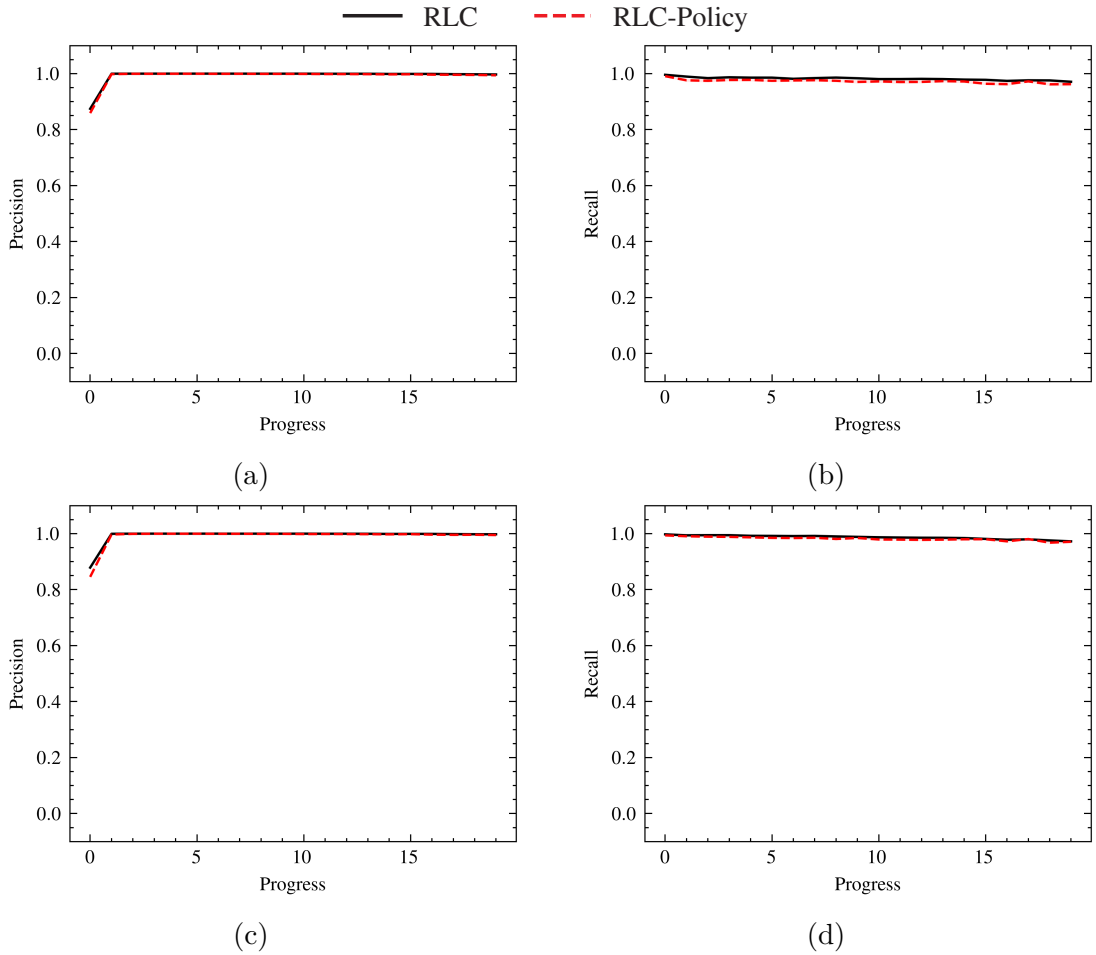


Fig. 5.28 The *Precision* and *Recall* in the manipulation task (Q-learning) applied EST-Action: (a)(b) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{1, 1\})$. (c)(d) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$.

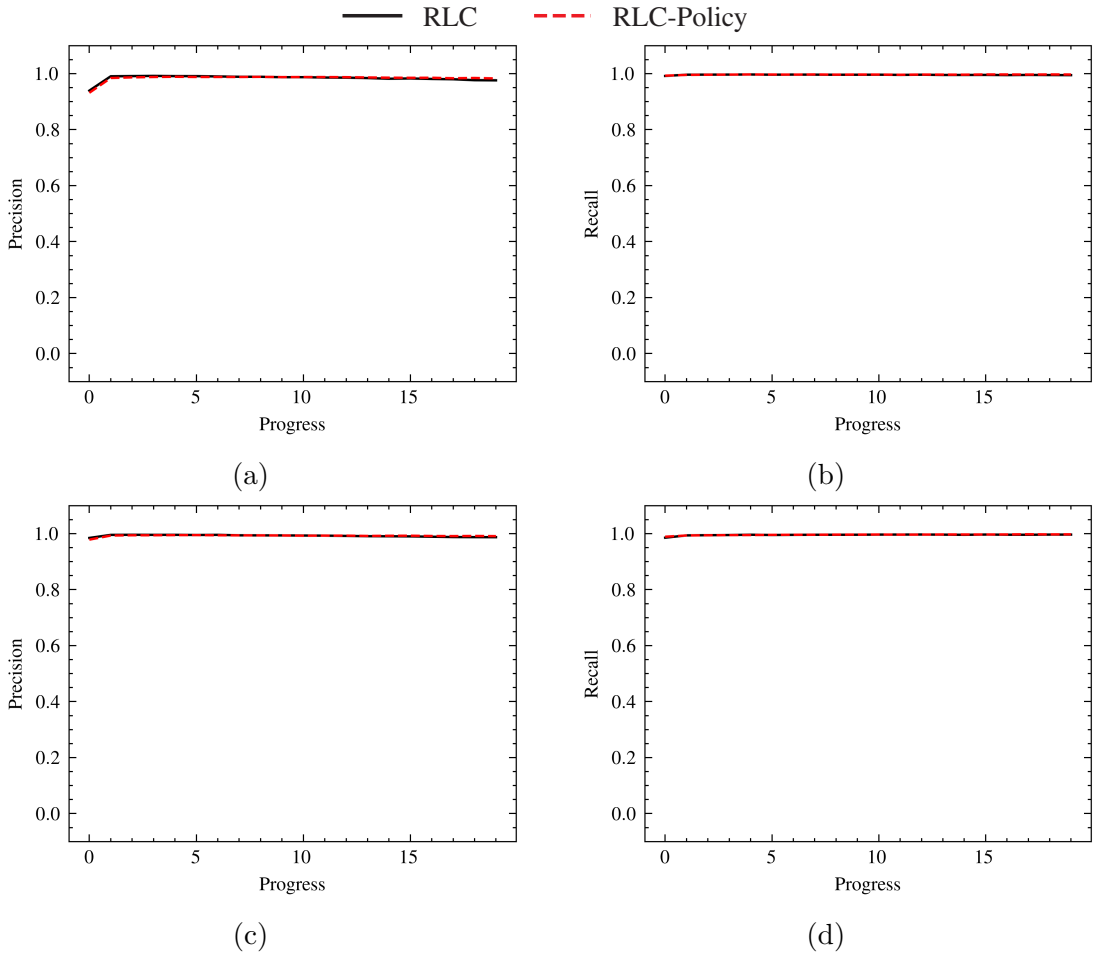


Fig. 5.29 The *Precision* and *Recall* in the manipulation task (Q-learning) applied EST-State-Policy: (a)(b) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{1, 1\})$. (c)(d) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$.

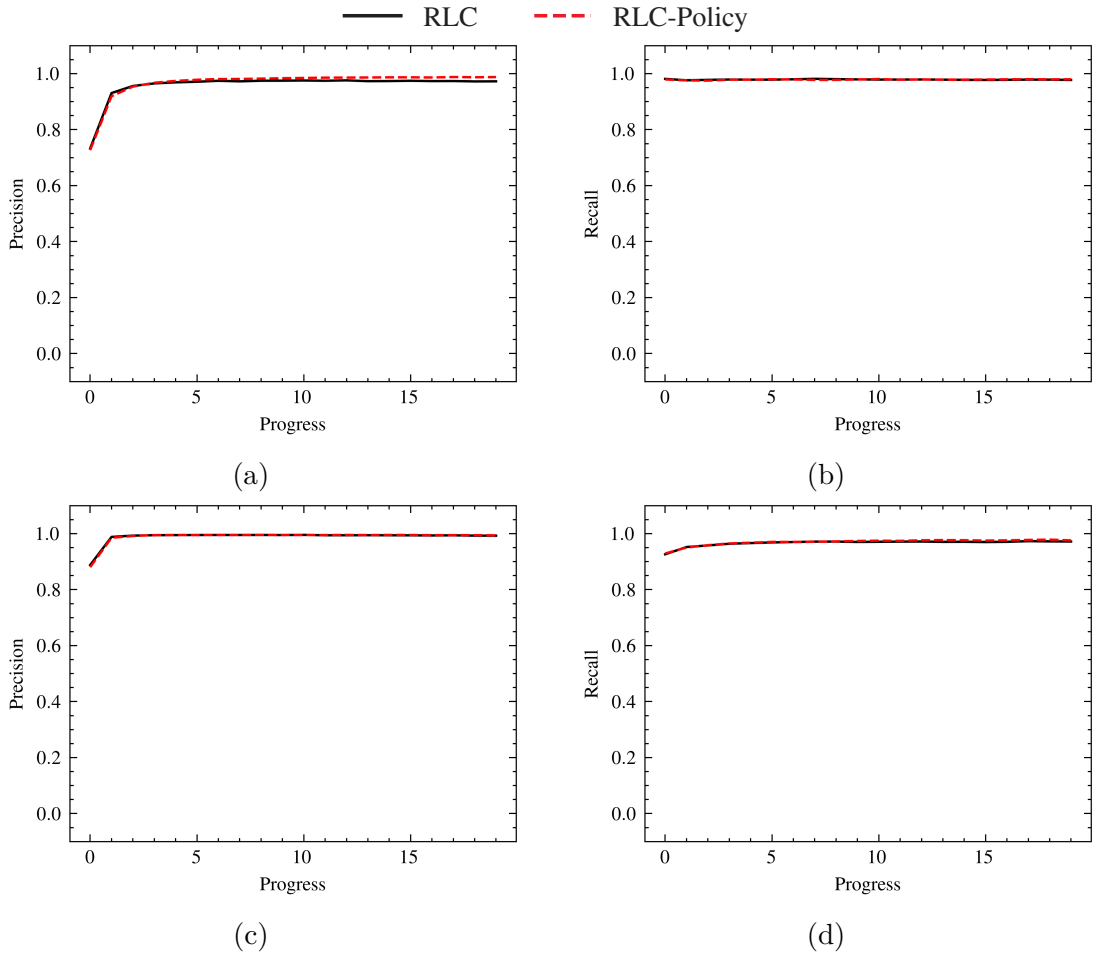


Fig. 5.30 The *Precision* and *Recall* in the manipulation task (Q-learning) applied EST-Policy: (a)(b) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{1, 1\})$. (c)(d) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$.

Having assessed *Precision* and *Recall*, now allow us to evaluate *Accuracy* and *Hamming Loss*. Figure 5.31, Figure 5.32 and Figure 5.33 present *Accuracy* and *Hamming Loss* achieved by the proposed estimation strategies. The x-axis of all these subplots maintains its representation of the 20 learning intervals. We can still find that *Accuracy* increases, while *Hamming Loss* decreases. We can continue to draw the similar conclusion as *Precision* and *Recall*. The *Accuracy* and *Hamming Loss* values of EST-State-Policy changes more smoothly in the initial stage than that of EST-Action and EST-Policy. There are still no great differences in *Accuracy* and *Hamming Loss* between RLC and RLC-Policy.

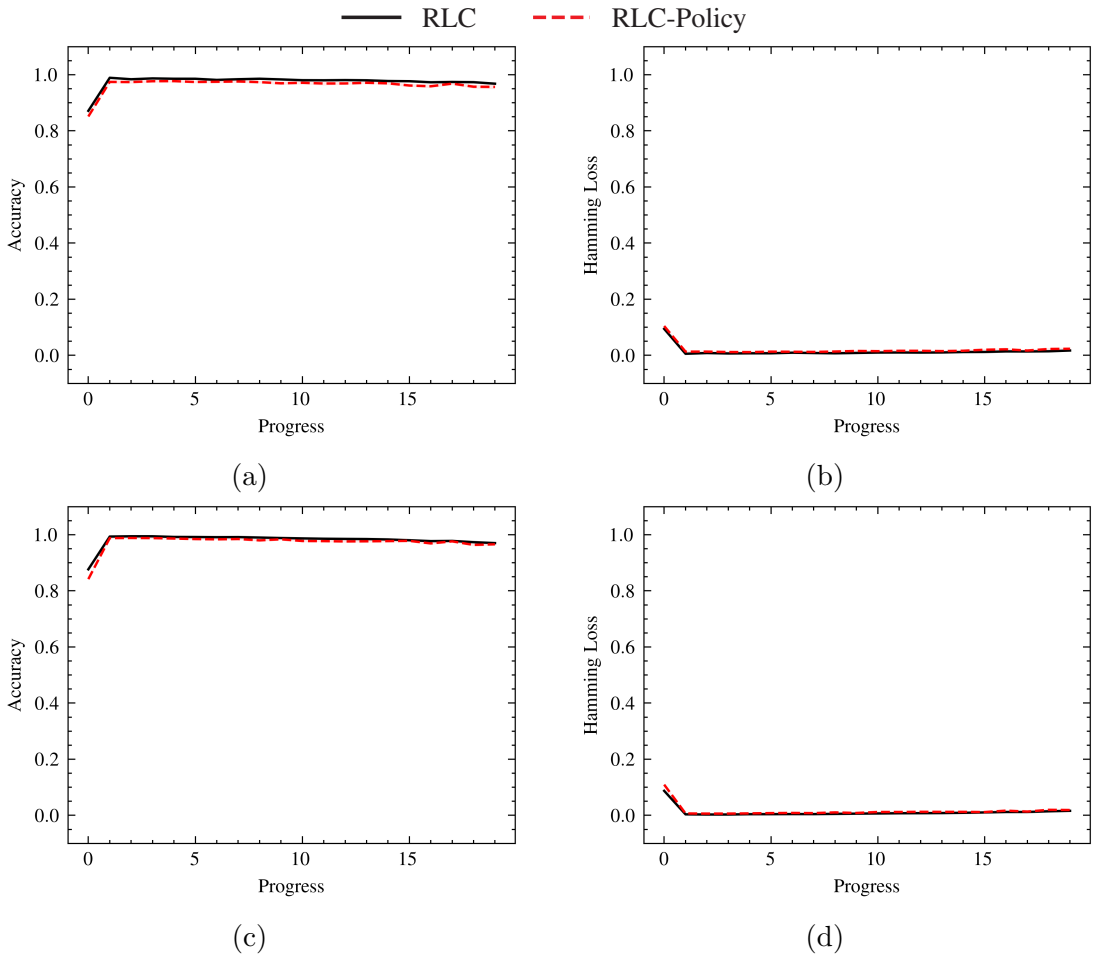


Fig. 5.31 The *Accuracy* and *Hamming Loss* in the manipulation task (Q-learning) applied EST-Action: (a)(b) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{1, 1\})$. (c)(d) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$.

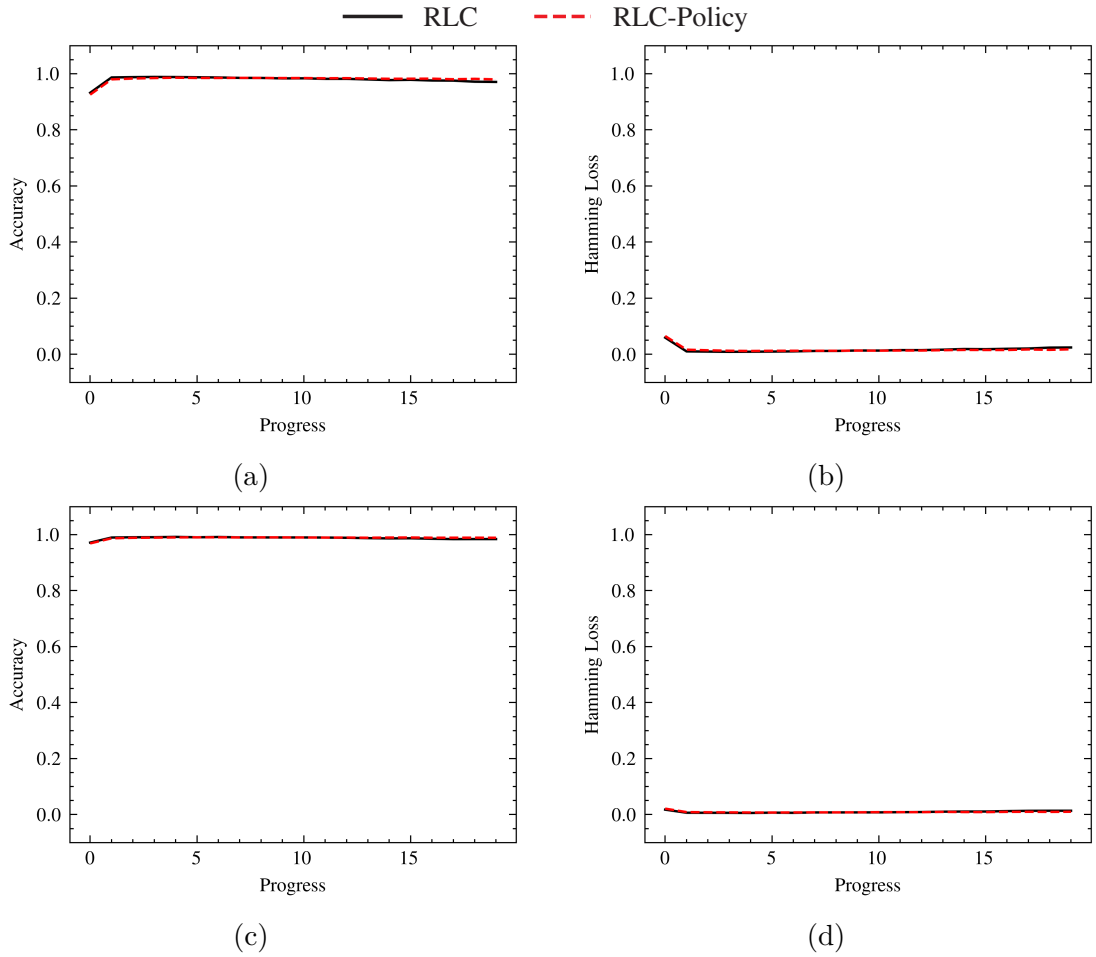


Fig. 5.32 The *Accuracy* and *Hamming Loss* in the manipulation task (Q-learning) applied EST-State-Policy: (a)(b) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{1, 1\})$. (c)(d) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$.

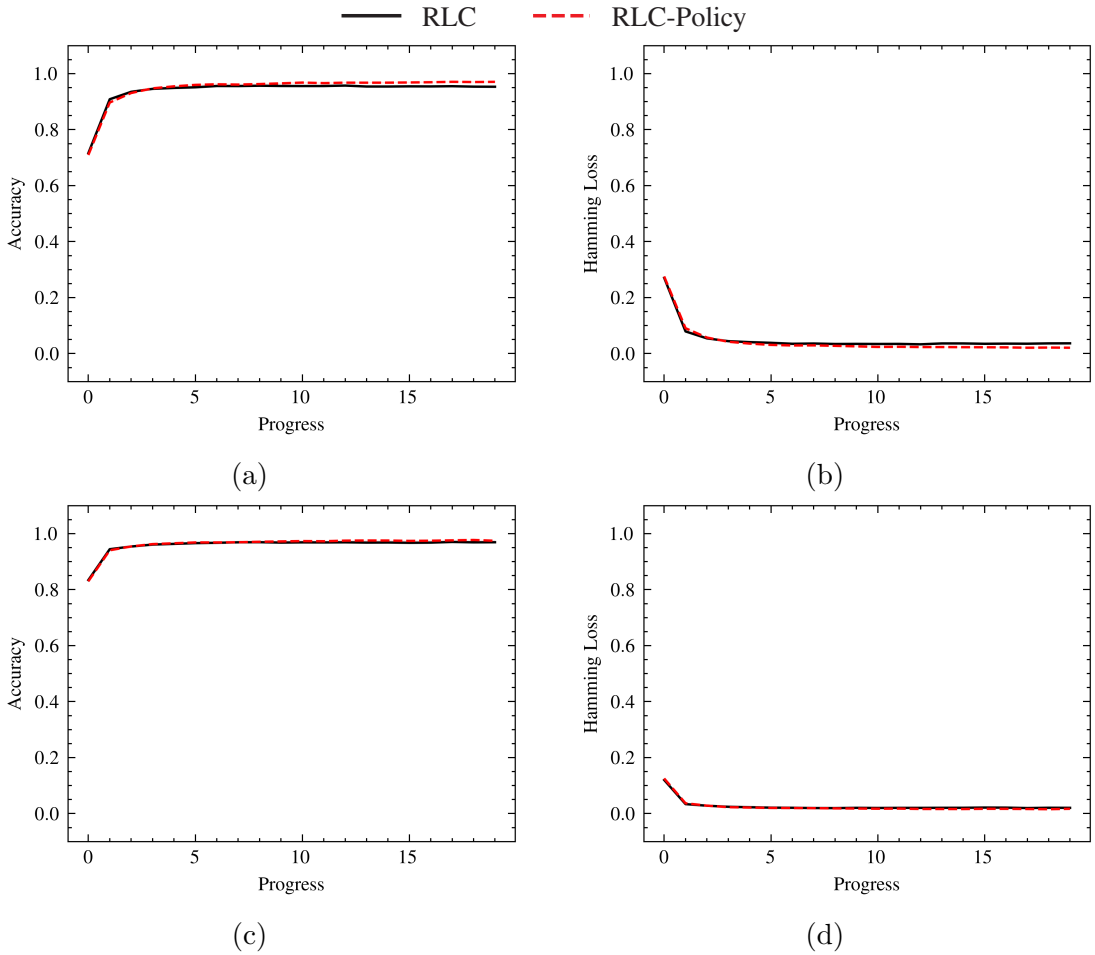


Fig. 5.33 The *Accuracy* and *Hamming Loss* in the manipulation task (Q-learning) applied EST-Policy: (a)(b) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{1, 1\})$. (c)(d) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$.

The above evaluation metrics *Precision*, *Recall*, *Accuracy*, and *Hamming Loss* show that our framework can effectively estimate the probabilities of human capabilities in this manipulation task. Then, allow us to examine how capability probabilities change over time. Figure 5.34, Figure 5.35, and Figure 5.36 offer a visualization of the capability probability convergence employing different estimation strategies, where the black line represents the probability associated with `c_green`, and the red line represents the probability associated with `c_swap`. All agents in these figures were initialized with the pre-trained policy $\pi^*(a \mid s, random)$ and applied RLC-Policy. High prior probabilities and initial policy affect the value of capability probabilities in the initial stage. Nevertheless, all estimates in these subplots gradually converge to the correct side and fluctuate on the proper side.

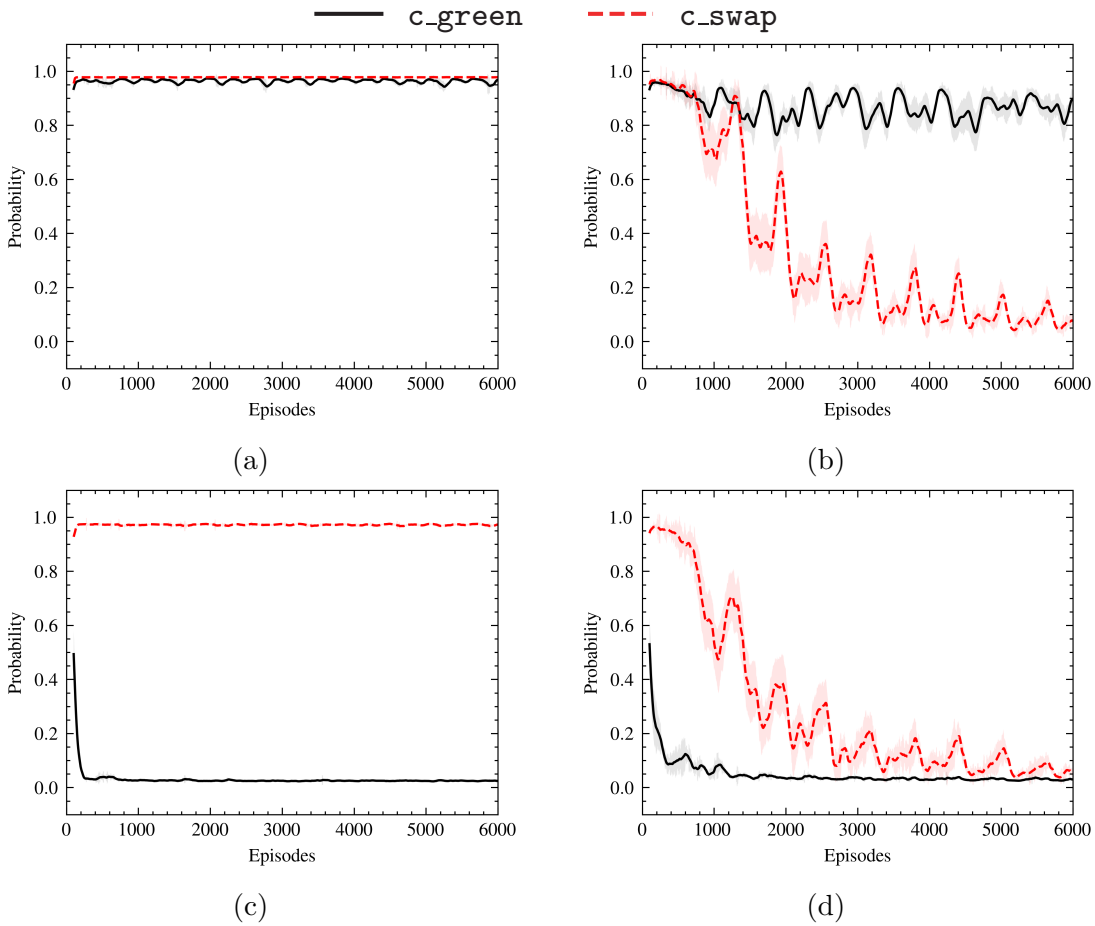


Fig. 5.34 The estimates of capabilities in the manipulation task (Q-learning) applied EST-Action and RLC-Policy, where all agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$: (a) The true collaborator's capability set was $\{\texttt{c\_green} = 1, \texttt{c\_swap} = 1\}$. (b) The true collaborator's capability set was $\{\texttt{c\_green} = 1, \texttt{c\_swap} = 0\}$. (c) The true collaborator's capability set was $\{\texttt{c\_green} = 0, \texttt{c\_swap} = 1\}$. (d) The true collaborator's capability set was $\{\texttt{c\_green} = 0, \texttt{c\_swap} = 0\}$.

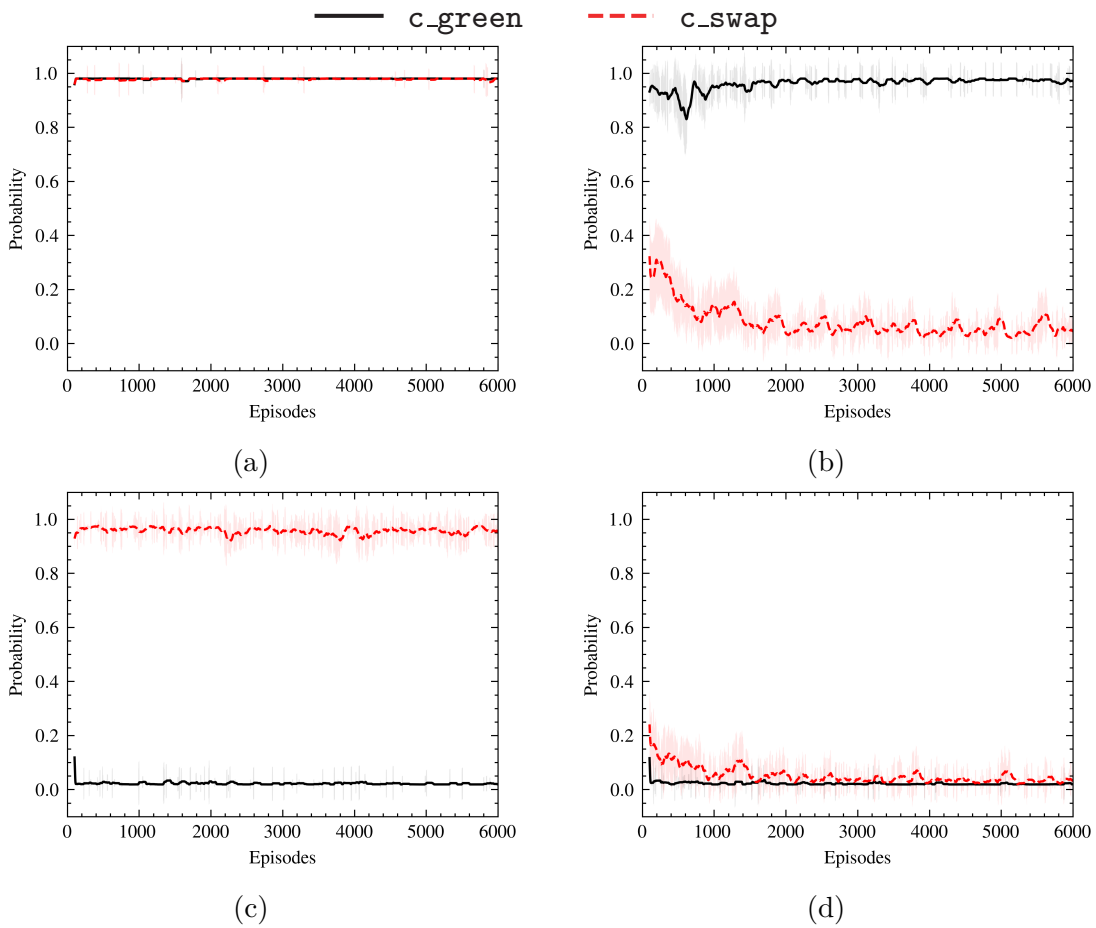


Fig. 5.35 The estimates of capabilities in the manipulation task (Q-learning) applied EST-State-Policy and RLC-Policy, where all agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$: (a) The true collaborator's capability set was $\{\texttt{c\_green} = 1, \texttt{c\_swap} = 1\}$. (b) The true collaborator's capability set was $\{\texttt{c\_green} = 1, \texttt{c\_swap} = 0\}$. (c) The true collaborator's capability set was $\{\texttt{c\_green} = 0, \texttt{c\_swap} = 1\}$. (d) The true collaborator's capability set was $\{\texttt{c\_green} = 0, \texttt{c\_swap} = 0\}$.

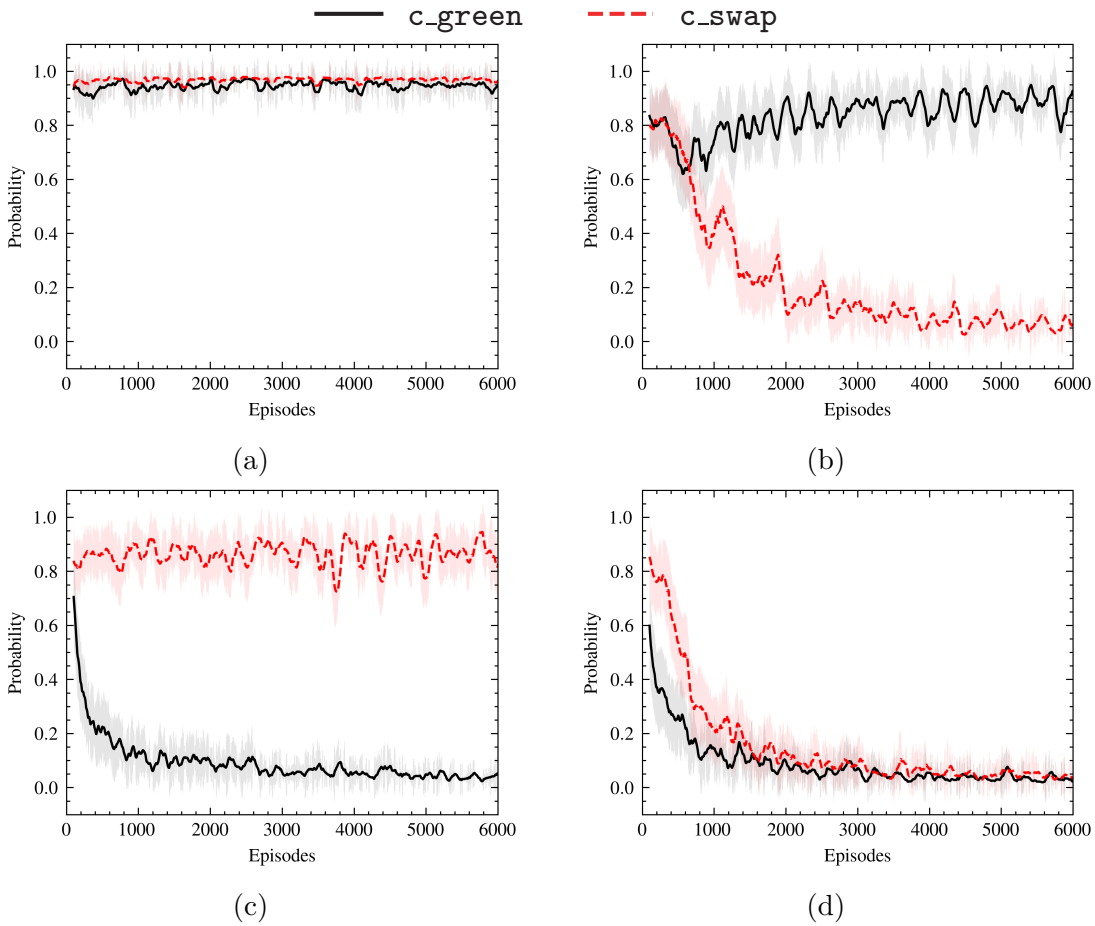


Fig. 5.36 The estimates of capabilities in the manipulation task (Q-learning) applied EST-Policy and RLC-Policy, where all agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$: (a) The true collaborator's capability set was $\{\texttt{c\_green} = 1, \texttt{c\_swap} = 1\}$. (b) The true collaborator's capability set was $\{\texttt{c\_green} = 1, \texttt{c\_swap} = 0\}$. (c) The true collaborator's capability set was $\{\texttt{c\_green} = 0, \texttt{c\_swap} = 1\}$. (d) The true collaborator's capability set was $\{\texttt{c\_green} = 0, \texttt{c\_swap} = 0\}$.

We continued investigating changes in capability probabilities over time in our DQN implementation with a sensitivity analysis regarding the initial policy selection. Figure 5.37 and Figure 5.38 offer a visualization of the capability probability convergence using EST-Policy with different initial policies. These two figures show the average result across configurations in DQN deployment, accompanied by 95% confidence intervals, calculated based on 20 trials.

The agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{1, 1\})$ in the Figure 5.37 and $\pi^*(a \mid s, random)$ in the Figure 5.38. Relying on the robust generalization ability of DQN, the capability probabilities converge to the correct side with slight fluctuation after only a small number of episodes. Nonetheless, high prior probabilities and the initial policy influence the initial value of capability probabilities.

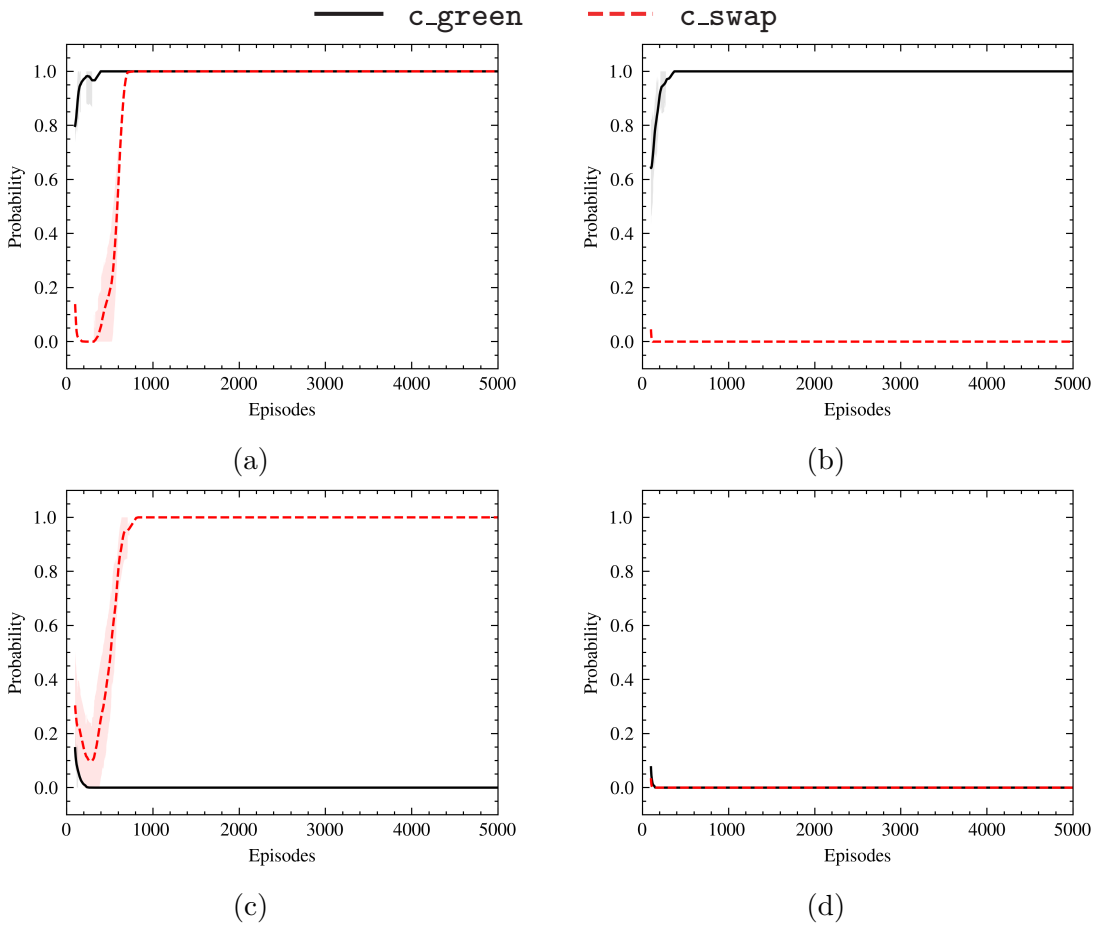


Fig. 5.37 The estimates of capabilities in the manipulation task (DQN) applied EST-Policy, where all agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{1, 1\})$: (a) The true collaborator's capability set was $\{\texttt{c\_green} = 1, \texttt{c\_swap} = 1\}$. (b) The true collaborator's capability set was $\{\texttt{c\_green} = 1, \texttt{c\_swap} = 0\}$. (c) The true collaborator's capability set was $\{\texttt{c\_green} = 0, \texttt{c\_swap} = 1\}$. (d) The true collaborator's capability set was $\{\texttt{c\_green} = 0, \texttt{c\_swap} = 0\}$.

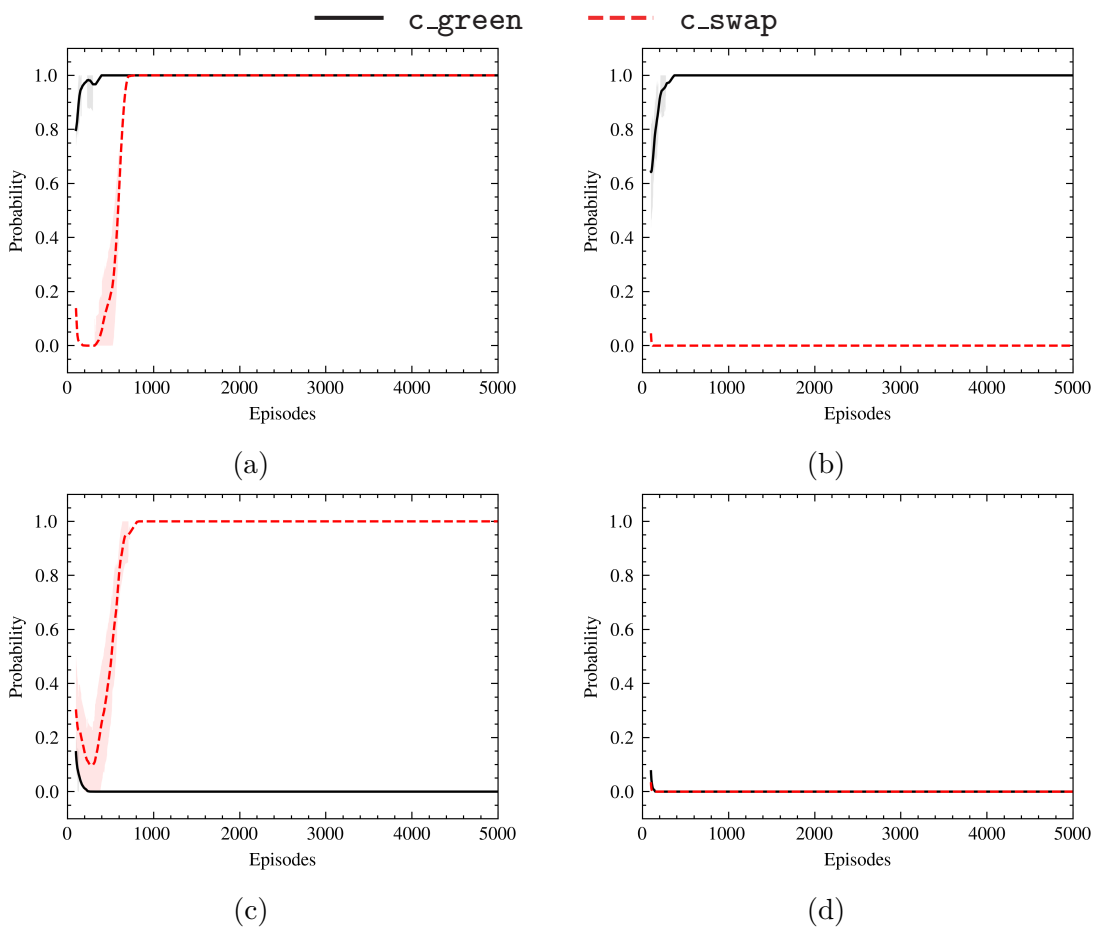


Fig. 5.38 The estimates of capabilities in the manipulation task (DQN) applied EST-Policy, where all agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$: (a) The true collaborator's capability set was $\{\texttt{c\_green} = 1, \texttt{c\_swap} = 1\}$. (b) The true collaborator's capability set was $\{\texttt{c\_green} = 1, \texttt{c\_swap} = 0\}$. (c) The true collaborator's capability set was $\{\texttt{c\_green} = 0, \texttt{c\_swap} = 1\}$. (d) The true collaborator's capability set was $\{\texttt{c\_green} = 0, \texttt{c\_swap} = 0\}$.

5.3.8 RL Performance Evaluation

In this subsection, we continue to run an ablation study over the various configurations of our framework and focus on whether RL agents applying our framework have more promising learning performance than the normal RL.

We first assessed our framework's RL performance through the Q-learning results. The Figures 5.39, 5.40 and 5.41 shows the *Return* and steps per episode of the normal RL, RLC and RLC-Policy. For capability estimation, the agents employed EST-Action in Figure 5.39, EST-State-Policy in Figure 5.40, and EST-Policy in Figure 5.41. These figures are accompanied by 95% confidence intervals, computed from 30 separate trial runs.

Unlike the previous navigation experiment with fixed initial and goal locations, the environment of this manipulation task is dynamic due to randomly generated initial layouts, which is reflected in the fact that all agents require more episodes to improve their policies.

Firstly, let us consider the impact of different exploration methods on RL performance by comparing the difference between the three lines in each subplot, where black is for normal RL, red is for RLC, and blue is for RLC policy. In analyzing the comparative effects between RL and RLC, one can note a marginal advantage of RLC over RL. The advantage may appear minimal, given the numerical proximity of immediate rewards associated with various actions. Nonetheless, the slight edge of RLC is relevant and contributes to learning. Meanwhile, RLC-Policy demonstrated a more remarkable advantage within this dynamic environment, consistently achieving higher returns while taking fewer steps than RLC and the normal RL. This observation suggests that, within the context of this experiment, the insights obtained by the agent from pre-trained policies based on capability estimates prove more conducive to generating user-adaptive behavior than simply applying the reduced action space.

Afterward, allow us perform a sensitivity analysis on different initial policies by analyzing the differences between pairs of subplot pairs: $\{(a), (b)\}$ and $\{(c), (d)\}$ in each figure. The convergence of online RL remains unaffected by the initial policy choice. In this dynamic environment, the two initial policies did not result in substantial differences in RL performance.

Lastly, by comparing these three figures comprehensively, we can see that the shift in capability estimation strategies maintains a minimal impact on the RL performance of this manipulation task. Because all three estimation strategies yield correct estimates.

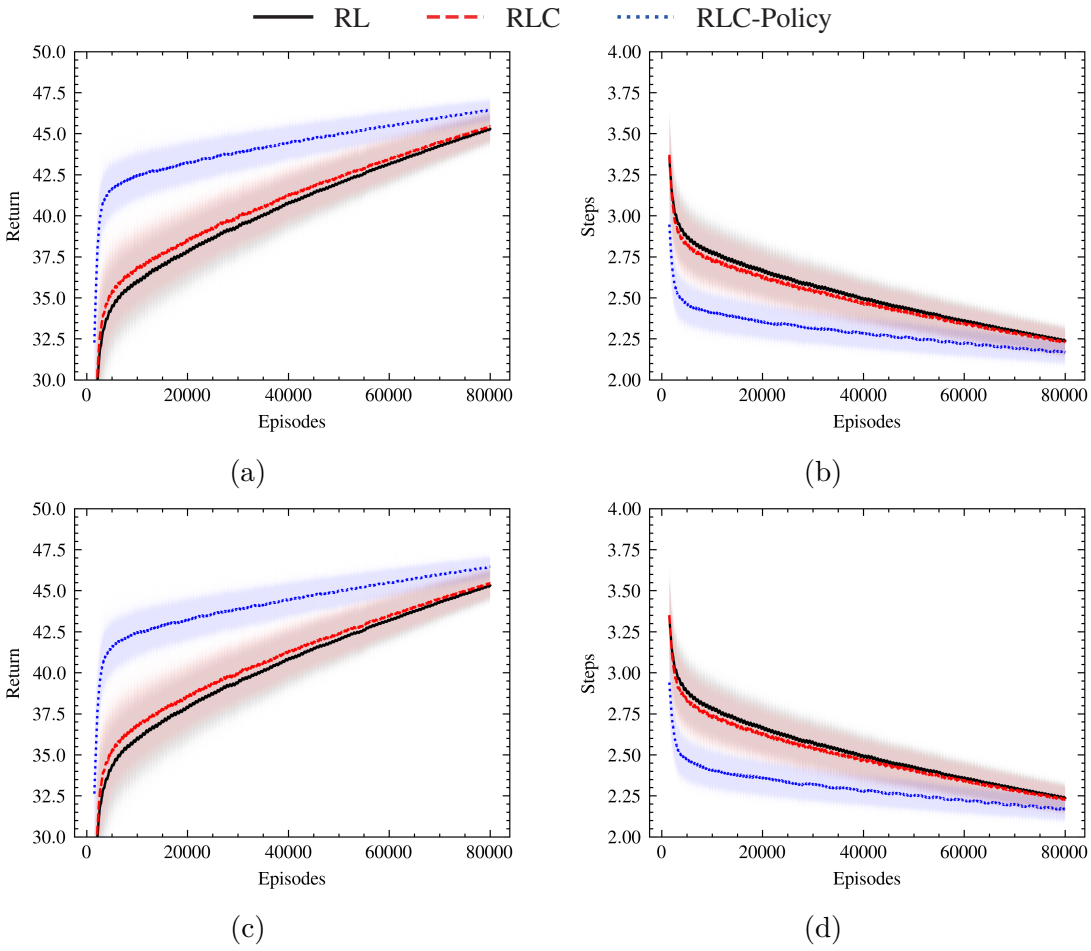


Fig. 5.39 The *Return* and the number of steps per episode in the manipulation task (Q-learning) applied EST-Action: (a)(b) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{1, 1\})$. (c)(d) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$.

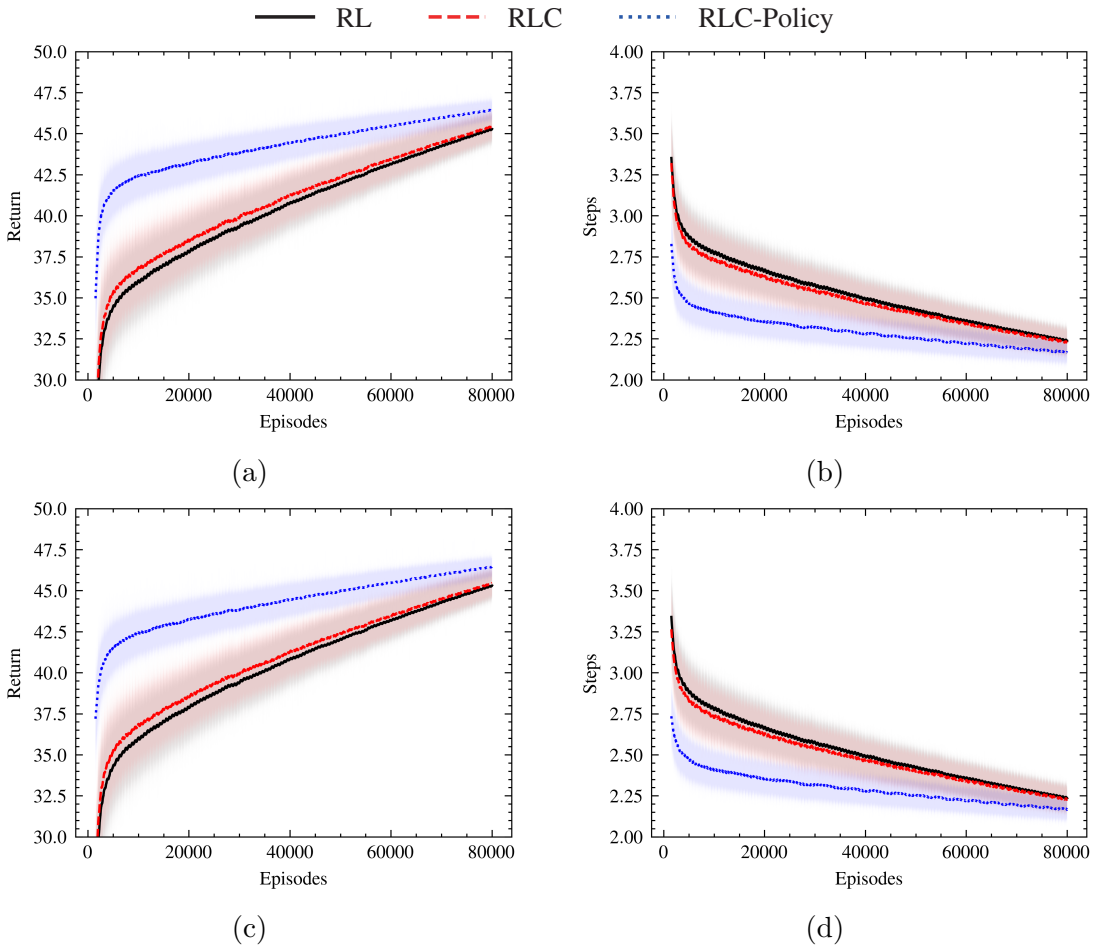


Fig. 5.40 The *Return* and the number of steps per episode in the manipulation task (Q-learning) applied EST-State-Policy: (a)(b) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{1, 1\})$. (c)(d) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$.

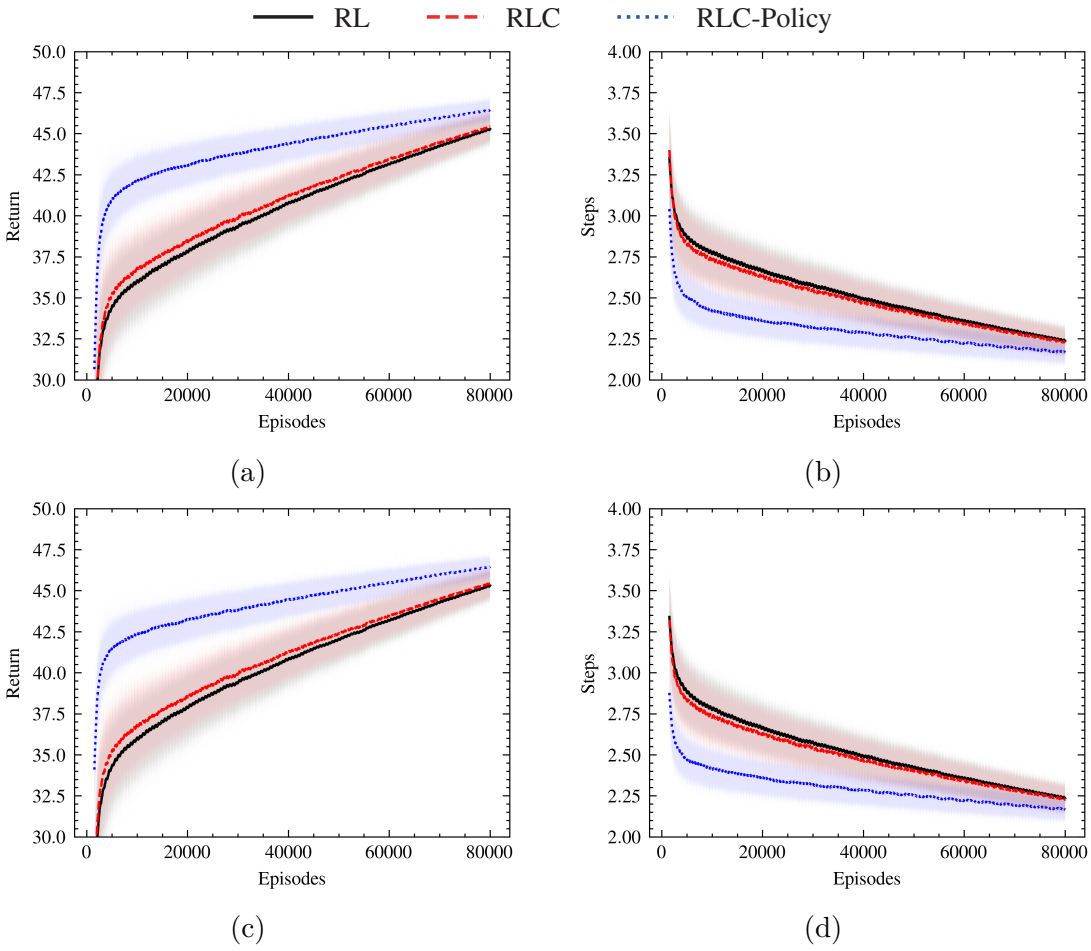


Fig. 5.41 The *Return* and the number of steps per episode in the manipulation task (Q-learning) applied EST-Policy: (a)(b) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{1, 1\})$. (c)(d) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$.

Moving forward, let us check the results of DQN implementation. Figure 5.42 shows *Return* and the number of steps taken per episode of the normal RL, RLC, and RLC-Policy. All subplots in this figure are accompanied by 95% confidence intervals, computed from 20 separate trial runs. We can continue to conclude that RLC and RLC-Policy yield higher *Return* with fewer steps than normal RL. RLC-Policy tends to deliver even better enhancements. Furthermore, while DQN exhibited a faster convergence rate than Q-learning in the static navigation task, this rate increase was even more pronounced for DQN in this dynamic manipulation task.

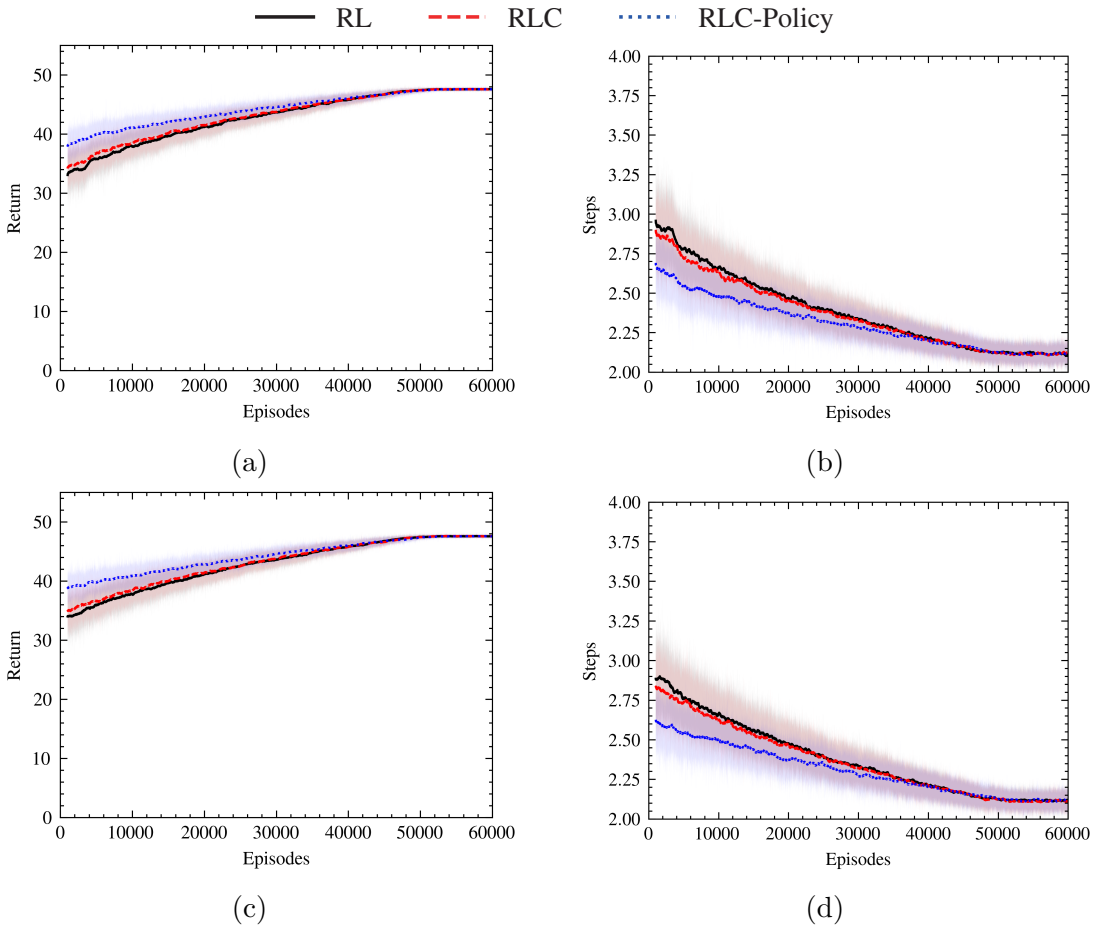


Fig. 5.42 The *Return* and the number of steps per episode in the manipulation task (DQN) applied EST-Policy: (a)(b) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, \{1, 1\})$. (c)(d) Agents were initialized with the pre-trained policy $\pi^*(a \mid s, random)$.

5.3.9 Summary

This section presented a more complex robot manipulation task than the robot navigation task, outlining the parameters associated with Q-learning and DQN implementations. The experimental results followed it. We carried out ablation studies focusing on two dimensions: estimation performance and RL performance. We performed a sensitivity analysis on the selection of initial policies.

5.4 Scavenger Hunt Game

5.4.1 Introduction

In this section, we present a collaborative variant of the Scavenger Hunt game [63], a benchmark for testing autonomous robots in the real-world settings. This task aims to find some objects in several rooms, and in our variant, the robot and a human collaborator work together.

At the beginning of each episode, a referee program generates a layout of objects based on a predefined distribution. The referee assistant then places the objects accordingly in the environment. With knowledge of the object distribution but not the exact object locations, the robot always guides the way during the search. When both the robot and a fully capable human collaborator are present, the most effective strategy involves the robot leading the way and the person identifying objects. In case of need, the robot can slow its pace during navigation and rely on its own vision detection if enough evidence shows that the human cannot assist. Additionally, the robot can explicitly ask the collaborator about the presence of an object during the game and decide whether to trust his response. This collaborative approach allows the robot to leverage both its own abilities and the assistance of the collaborator, resulting in an efficient and user-adaptive system for the Scavenger Hunt task.

In this thesis, the experiment was conducted with the researcher serving as the sole subject, designed to show the technical applicability of our framework in real-world scenarios. The design of this experiment is entirely feasible with real volunteers. However, recruiting at that time would no doubt be impeded by the COVID-19 pandemic.

5.4.2 MDP Settings

The state space consists of the position of the robot base, the current navigation goal, the target object, the latest answer from the person, the robot's distance from the person, and the position of the objects found. The action space consists of `drive_slow`, `drive_fast`, `ask`, `inform`, `accept`, and `reject`. `drive_slow` represents the robot's mobile base moving to a waypoint at low speed, while `drive_fast` is at high speed. `ask` represents the robot asking the person if he can find the target object around the current location. Then, the robot has two actions: `accept` and `reject`. `accept` represents that the robot trusts and takes the human's answer. `reject` represents that the answer is rejected, and the robot plans to scan the area by itself. Besides

asking for human responses, the robot has an alternative action `inform`. This action represents the robot expecting the person's answer to be wrong and sticking to the robot's observations. Rather than the reject way: explicitly asking the person, rejecting the answer, and then scanning around, it is considerably time-efficient to perform the `inform` action: scanning the area directly and telling the person the robot's answer.

The reward function is as follows. Denoting with d the human-robot distance, and with $d_{max} = 4$ the desired maximum distance, The rewards are assigned to the agent based on the rules listed below, otherwise the reward is 0. The task aims to encourage human-robot collaborations instead of minimizing completion time arbitrarily. Thus, the reward function prioritizes actions that involve discussing the location of objects with the collaborator over providing answers directly by the robot even if the latter takes less time than the former:

- If $d \leq d_{max}$: $R_a(\texttt{drive\_fast}) = -1$, $R_a(\texttt{drive\_slow}) = -2$.
- If $d > d_{max}$: $R_a(\texttt{drive\_fast}) = -20$, $R_a(\texttt{drive\_slow}) = -10$.
- If the robot correctly identifies the location of an object that is in the room: $R_a(\texttt{accept}) = 80$, $R_a(\texttt{inform}) = 60$, $R_a(\texttt{reject}) = 40$.
- If the robot does not return a false positive, for an object that is not, in fact, in the room (otherwise the robot may just guess in every room): $R_a(\texttt{accept}) = 30$, $R_a(\texttt{inform}) = 20$, $R_a(\texttt{reject}) = 10$.

5.4.3 Human Capabilities

Four human capabilities `c_fast`, `c_color`, `c_sight`, and `c_sound` are considered in this task. The `c_fast` capability represents that the person can walk fast; `c_color` represents that the person can discriminate red and orange; `c_sight` represents that the person can recognize objects from far away; `c_sound` represents that the person can locate hidden objects that emit a sound.

5.4.4 Task Settings

There are six rooms to be searched in the Bragg building at the University of Leeds. Two of them are shown in Figure 5.43. Four target objects `red_ball`, `chickpeas`, `breadsticks` and `speaker` are shown in Figure 5.44. A blue ball or orange ball may also be placed where the `red_ball` may be, requiring `c_color` to be discriminated by the collaborator. The canned `chickpeas` can be found anywhere. The box of `breadsticks`

will be placed far away from the center of the room. Thus, the collaborator requires `c_sight` to see it. The speaker will be turned on to play music and placed in a position invisible to the robot and the collaborator, requiring `c_sound` to be located.

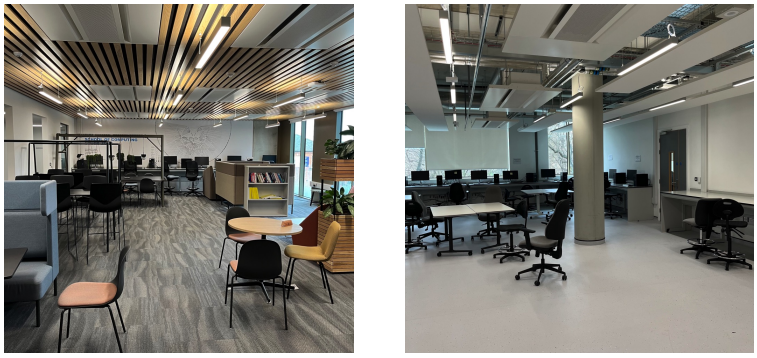


Fig. 5.43 Examples of searching rooms in the Bragg building at the University of Leeds.

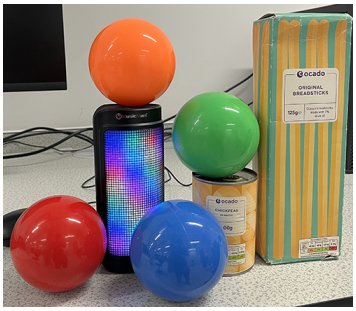


Fig. 5.44 The objects being searched in the Scavenger Hunt Game.

5.4.5 Human-Robot Distance Estimation

To satisfy the requirement of human-robot distance measurement used in the state space, we investigated two indoor positioning technologies: Ultra Wide Band (UWB) [4] and Bluetooth [61]. UWB stands out for its robust resistance to interference and its capacity to attain centimeter-level positioning accuracy. On the other hand, Bluetooth-based Indoor Positioning, while less resilient to interference, boasts widespread support in mobile devices, and the equipment required: Bluetooth beacons are inexpensive and easy to deploy. We chose the Bluetooth technique in the end.

Given that our experiment takes place in the public area of the Bragg building, the accuracy of Bluetooth-based indoor localization is easily affected by signal refraction and obstruction. Thus, we deployed four Bluetooth beacons on the torso of the TIAGo robot for error correction, as shown in Figure 5.45.

Fig. 5.45 The Bluetooth beacons deployed on TIAGo.

The principle of Bluetooth Indoor Positioning relies on the positive correlation between the Received Signal Strength Indicator (RSSI) of the Bluetooth beacon measured by the transceiver device and the distance between them. In ideal conditions, the distance d between the transceiver device and the Bluetooth beacon can be estimated using Equation (5.5)[2], where $RSSI_{TX}$ represents the Transmission Power, $RSSI$ is the signal strength, and n is an empirical value of path-loss exponent, ranges from 2 to 4:

$$d = 10^{\frac{RSSI - RSSI_{TX}}{-10n}}. \tag{5.5}$$

Algorithm 5.6 shows the process of RSSI updating. The RSSI of all beacons is obtained, represented by the vector $\mathbf{r}$, at line 2. In the real environment, the signal strength value can be affected by electromagnetic interference, such as WIFI and the cellular network. To mitigate the noise impact and ensure more robust measurements, we employ a Gaussian filter (line 5) to smooth the RSSI values. This filtering process involves applying a one-dimensional Gaussian distribution (shown in Equation (5.6)) with a standard deviation $\sigma = 1$, and a kernel size of 9:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}. \tag{5.6}$$

Algorithm 4: RSSI Updating

Input: m: The max queue size of smoothed RSSI values.
σ: The standard deviation of Gaussian filter.
k: The kernel size of Gaussian filter.

$\mathbf{r}_s \longleftarrow empty\_dqueue(maxlen = m)$;
while *task is running* **do**
  $\mathbf{r} \longleftarrow result\_from\_beacons()$;
  $\mathbf{r}' \longleftarrow \mathbf{r} \bigcup \mathbf{r}_s$;
  $\mathbf{r}' \longleftarrow gaussian\_filter1d(\mathbf{r}', \sigma, k)$;
  $r_{new} \longleftarrow$ the last value of $\mathbf{r}'$;
  Append r_{new} to $\mathbf{r}_s$;
  Publish r_{new} as the latest RSSI value.
end

5.4.6 Mobile Software

The collaborator carries a smartphone with software to answer the robot's questions, providing two options `Yes` and `No`. At the same time, the software obtains the RSSI of the Bluetooth beacons through the Application Programming Interface (API) of the mobile phone, and uploads it to the server in real-time. The robot also carries a tablet reporting the text of the robot's words, allowing hearing-impaired people to answer and interact. Figure 5.46 shows the whole structure of this task.

5.4.7 Pre-training in Simulation

Although the deployment of this task was carried out in a real environment, considering that there are a total of 16 capability combinations, pre-training in a real environment is impractical. Therefore, pre-training for this task was performed in the simulator. Our simulator environment incorporated the following features to enhance the robustness of the pre-training policies and facilitate smoother transfer to the real-world environment:

(i) There is a probability of 0.1 that the collaborator slowly walks even if he has the capability `c_fast`.

(ii) The robot cannot directly see the expected target object when entering the room with a probability of 0.5 unless it performs a scanning.

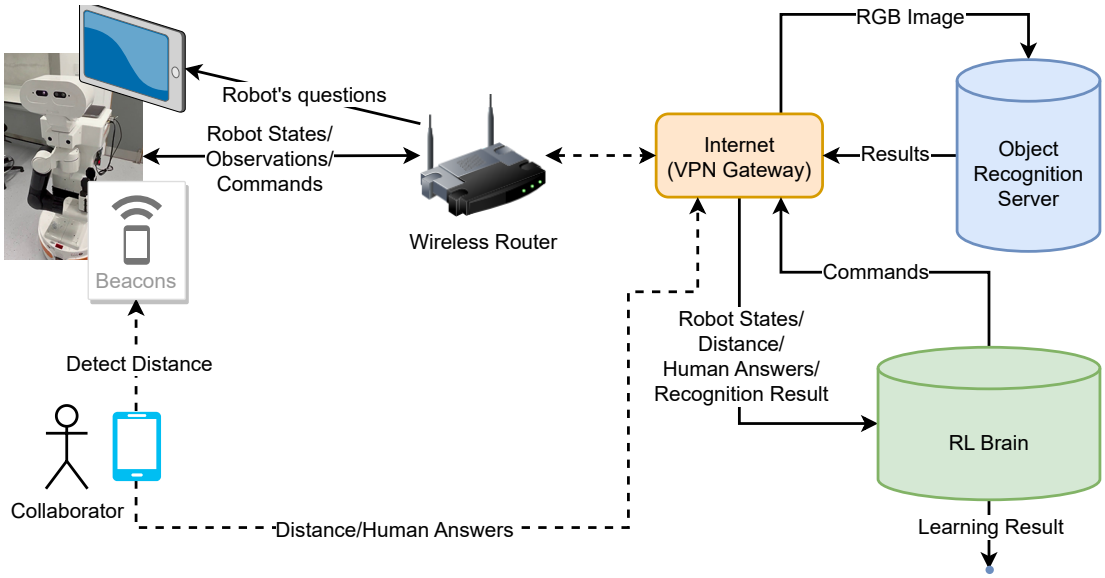


Fig. 5.46 The structure of the implementation in the Scavenger Hunt Game.

(iii) Even if scanning is performed, there is still a probability of 0.2 that the robot cannot recognize the target object.

(iv) There is a probability of 0.1 that the robot can observe the speaker without help from the collaborator.

(v) There is a probability of 0.2 that the robot's scanning will produce a false positive result.

(vi) There is a probability of 0.1 that the collaborator will give the robot an answer inconsistent with the truth.

5.4.8 Experimental Setup

We only implemented Q-learning in this domain. The experiment started with pre-training, resulting in $|2^{\mathbf{c}}| = 16$ pre-trained policies and one extra pre-trained policy for random capabilities. The pre-trained policy $\pi^*(a \mid s, random)$ is the initial policy. There are a total of 16 capability combinations in this task, and we selected the following three combinations in the deployment: (i) {`c_color`,`c_sound`}, (ii) {`c_fast`,`c_sight`,`c_color`}, (iii) {`c_sight`,`c_color`}. For each initial policy, the RL agents were deployed separately with collaborators of these three capability combinations. We tested all combinations of {EST-Policy} × {RL(normal), RLC-Policy}

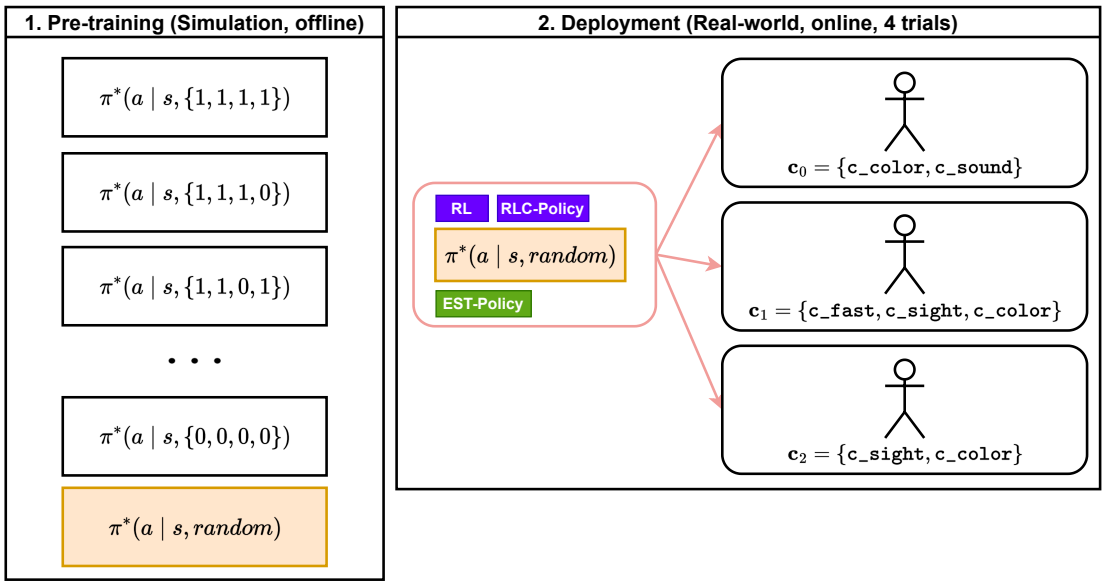


Fig. 5.47 The experimental setup of the Scavenger Hunt game (Q-learning), where the yellow rectangles represent the pre-trained policies used in the initialization of deployments.

separately in the deployment. Consequently, $1 \times 3 \times 1 \times 2 = 6$ deployment configurations. Figure 5.47 shows the overview of the experimental setup.

In pre-training, the agents were trained over 5,000,000 episodes for each capability combination and the random capabilities in the simulator. The learning rate $\alpha = 0.1$, the discount factor $\gamma = 0.95$. The maximum number of steps in a single episode was 200. The prior probability of combination $\{\texttt{c\_fast} = 1, \texttt{c\_sight} = 1, \texttt{c\_color} = 1, \texttt{c\_sound} = 1\}$ was 0.8, and for the other combinations was 0.0133.

The agents ran 20 episodes for each configuration. The ϵ of ϵ-greedy decayed linearly from 0.4. The learning rate $\alpha = 0.3$, the discount factor $\gamma = 0.95$. The maximum number of steps in a single episode was 100. The capability thresholds $\mathbf{d} = 0.5$. The Softmax temperature $\tau = 0.3$. The batch size of trajectory $l = 10$. The $\kappa = 0.5$ and $\kappa_{pre} = 0$. We chose the agent applying EST-Policy and RLC-Policy to compare with the normal RL. Thus, there is no sampling process in Figure 5.47. Overall, each configuration was evaluated for 4 trials. Each episode took approximately 20-30 minutes.

5.4.9 Estimation Performance Evaluation

Figure 5.48 shows the average probabilities of capabilities over time in the Scavenger Hunt Game, computed from 4 separate trial runs. All agents in these figures were initialized with the pre-trained policy $\pi^*(a \mid s, random)$, applied EST-Policy and RLC-Policy. The estimates fluctuated but were eventually correct in all configurations and almost always fluctuated on the correct side.

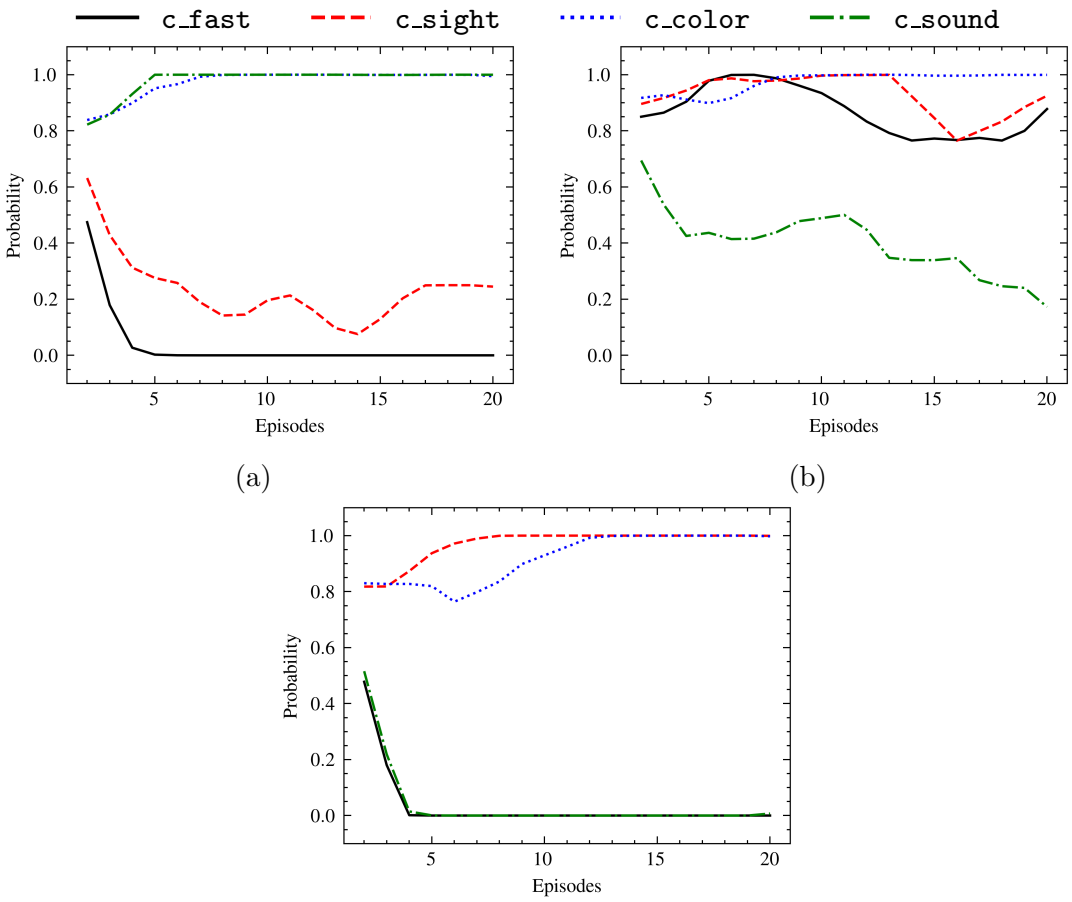


Fig. 5.48 The probabilities of capabilities in the Scavenger Hunt Game: (a) The true collaborator's capability set was {`c_color`,`c_sound`}. (b) The true collaborator's capability set was {`c_fast`,`c_sight`,`c_color`}. (c) The true collaborator's capability set was {`c_sight`,`c_color`}.

5.4.10 RL Performance Evaluation

In this experiment, the pre-trained policies were trained in the simulator. So, we focus on the agent's performance when the environment switches from the simulator to the real world. Figure 5.49 shows average *Return* across three tested capability combinations, which are computed from 4 separate trial runs. Our agent applying EST-Policy and RLC-Policy obtained higher *Return* than the normal RL. This result shows that the policy trained in the simulator combined with our framework can help the agent obtain better RL performance in this real-world human-robot collaborative task.

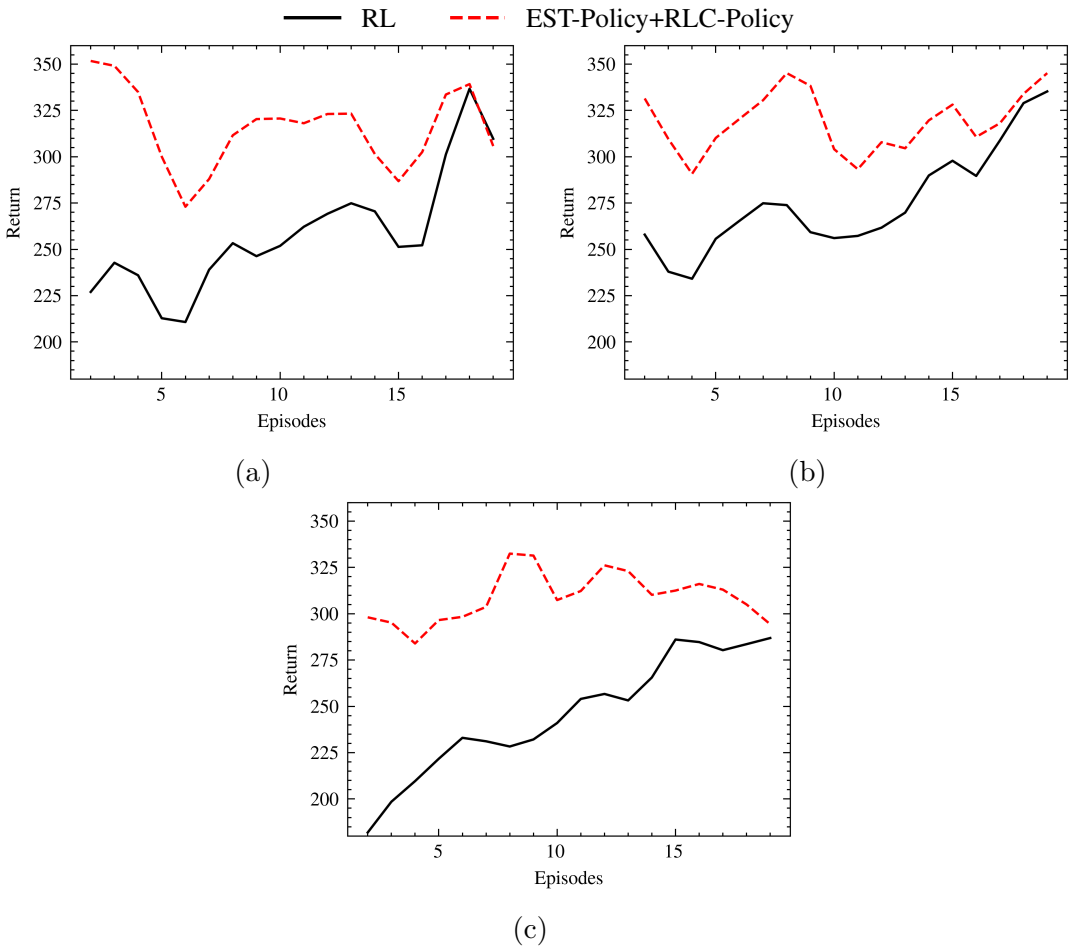


Fig. 5.49 The *Return* in the Scavenger Hunt Game: (a) The true collaborator's capability set was {`c_color`,`c_sound`}. (b) The true collaborator's capability set was {`c_fast`,`c_sight`,`c_color`}. (c) The true collaborator's capability set was {`c_sight`,`c_color`}.

Chapter 6

Conclusion and Future Work

6.1 Results Summary

In the previous chapter, we present results from three collaborative tasks: two conducted within a simulator environment and one as a real-world experiment. The first navigation task operated in a simple, deterministic environment, where a robot-guided a collaborator from a fixed starting point to a fixed goal. In contrast, the second manipulation task introduced uncertainty. The layout of objects on the table and the chessboard was randomly generated, requiring human-robot collaborations to organize these objects. The third experiment operated in the real world, the collaborative "Scavenger Hunt Game," also unfolded within an uncertain setting. Target objects were randomly placed in several rooms, and the robot's objective was to work with the collaborator to locate all target objects. The pre-training for the Scavenger Hunt Game took place in the simulator, while deployment operated in the real world.

We conducted ablation study and sensitivity analysis in the above environments, and our primary focus was on addressing the following key questions:

(i) **The impact of exploration strategies based on capability estimation and pre-training policies on RL performance.** When capability estimates are correct, the agents applying RLC and RLC-Policy obtained higher *Return* and fewer episode steps than normal RL. Overall, RLC-Policy, combining online capability estimation and capability-guided exploration, performs better than RLC and normal RL.

(ii) **The impact of different initial policies on RL performance**: While the convergence of Reinforcement Learning remains invariant to the initial policy, the adoption of an appropriate initial policy can significantly accelerate the learning

process. The $\pi^*(a \mid s, \{1, \cdots, 1\})$ initial policy adopts more aggressive behaviors. It can rapidly react when receiving a strong signal, such as negative rewards, to learn more suitable actions for the collaborator. The $\pi^*(a \mid s, \{0, \cdots, 0\})$ initial policy adopts more cautious behaviors to avoid task failure, but it may get stuck in local optima when the exploration is insufficient. The behavior of the initial policy $\pi^*(a \mid s, random)$ is between $\pi^*(a \mid s, \{1, \cdots, 1\})$ and $\pi^*(a \mid s, \{0, \cdots, 0\})$, and achieve relatively good performance. The selection of initial policies also impacts capability estimates, which we will continue to discuss in (vii).

(iii) **The impact of different capability estimation strategies on RL performance.** In our two simulator experiments, the impact of different estimation strategies on RL performance is not apparent. Because all three estimation strategies produced correct estimates. However, one must consider that these estimation strategies come with varying constraints and computational demands. Such constraints are given in our discussion (vi) related to estimation performance.

(iv) **How reliable are our capability estimation strategies as a side effect of Reinforcement Learning**? We employed metrics including *Accuracy*, *Precision*, *Recall*, and *Hamming Loss* in two simulation experiments to demonstrate that all capabilities probabilities can converge gradually to the correct values as policy improves, with reasonable fluctuations. Furthermore, we showed plots of capability probabilities over time in simulated and real experiments, providing additional evidence of the convergence of capability probabilities. However, disabling actions in Reinforcement Learning comes with both benefits and risks. We recommend not disabling actions during the exploitation process when capability estimates are unstable or unreliable in the initial stage.

(v) **The impact of prior probabilities on capability estimation and Reinforcement Learning performance.** In our navigation experiment, we conducted a sensitivity analysis focused on prior probabilities. The evidence demonstrates an observable influence of prior probabilities on the initial stage of capability estimation, which is similar to the initial policy made. However, this does not affect the convergence of capability probabilities. Meanwhile, when examining the impact on Reinforcement Learning metrics, the influence of these prior probabilities was minimal.

(vi) **Estimation performance differences of different estimation strategies.** Different estimation strategies demonstrated different estimation performances

Estimation Strategy	**State Space**	**Action Space**	**Sampling**
EST-Action	N/A	Discrete	Required for $A \sim$
EST-State-Policy	Discrete	N/A	Required for $S \sim$
EST-Policy	N/A	N/A	N/A

Table 6.1 The constraints of estimation strategies.

in our simulator experiments. Because EST-Action only considers the trajectory of actions, the estimates produced are not as stable as EST-State-Policy and EST-Policy. EST-State-Policy and EST-Policy may be more effective. However, the use of these strategies is subject to different constraints depending on the conditions of their computing methods. We introduced more details of these constraints in Section 4.3, and here we offer a summary in Table 6.1.

(vii) **The impact of different initial policies on capability estimation.** The initial policies are not directly introduced into the calculation of the capability estimation. Nevertheless, it plays a crucial role in shaping the agent's behavior during the early stage of learning. Therefore, the capability estimates at the initial stage of learning are correlated with the capability combination of the selected initial policy. For example: when the initial policy is $\pi^*(a \mid s, \{1, \cdots, 1\})$, the initial estimate is close to $\{1, \cdots, 1\}$. These initial estimates, rooted in prior knowledge, gradually converge towards the correct side as the agent's policy improves and relevant evidence accumulates over time.

(viii) **The performance of our method in Deep Reinforcement Learning.** DQN, used in our experiments as a Deep Reinforcement Learning algorithm, offers generalization abilities that Q-learning does not have and supports high-dimensional state spaces. Our method can seamlessly serve as middleware for DQN and Q-learning, simultaneously estimating capability probabilities and guiding agent exploration. It can be seen from the results of capability estimation that our method performed well in DQN. The convergence rate of capability probabilities in DQN is faster than in Q-learning.

6.2 Limitations and Future Work

6.2.1 Binary Capabilities

In this thesis, we treat human capabilities as binary variables. For instance, in the navigation experiment, we use `c_fast` $\in \{0, 1\}$ to represent if the collaborator can walk fast or slow. However, several human capabilities, like walking speed, are better represented in a continuous space.

We primarily use binary variables because we consider human capability estimation as a multi-label learning problem rather than a regression problem. Our goal is to determine a decision boundary. Binary labels clearly differentiate between the presence and absence of capabilities, which can be helpful for decision-making through the pre-condition model. Additionally, to acquire optimal policies used in the estimation, we have to perform the pre-training where binary human capability ensures a finite number of pre-trained policies.

Nevertheless, the binary representation disregards the variability within capabilities, including varying levels or degrees of expertise. A shift towards continuous capabilities could enable finer control and introduce more nuanced behaviors, representing a potential direction for future research. When dealing with continuous probability density functions, performing regression on the distribution would be necessary. This regression could require a large sample size to capture the variables' variability sufficiently.

6.2.2 The Human Objects of Real-World Experiment

We introduced a collaborative Scavenger Hunt Game designed for user engagement in real-world settings. We present this experiment as a technical implementation to demonstrate the feasibility of our framework in the real world. Moreover, when we organized that experiment, considering the COVID-19 pandemic and the associated health risks to participants and researchers, only the researcher stood as human subjects in this thesis. Future iterations of similar research could benefit from incorporating real users to further validate findings and enhance practical applications.

6.2.3 Optimized Multi-label Learning

In our capability estimation methods, we used the Label Powerset, as a high-order strategy, considering the correlations among all capability labels, generating a total of $|2^{\mathbf{c}}|$ capability combinations, which shows a significant limitation of the Label Powerset: When the size of capability set $\mathbf{c}$ grows, the number of capability combinations

increases exponentially, leading to high complexity and inefficient pre-training processes. Therefore, future work might involve introducing a high-order algorithm that can address the drawbacks of the Label Powerset, such as Random k-Labelsets [60]. Another way is to introduce Algorithm Adaptation Methods to solve the multi-label learning problem. This category of algorithms focuses on adapting existing learning algorithms to address multi-label learning problems effectively.

Another perspective is the feasibility of disregarding correlations between human capabilities. If such correlations can be overlooked, we can employ the first-order strategy in multi-label learning, which assesses the probability of each capability independently instead of their joint probability. There is no doubt that first-order strategies can simplify computational complexity. In this thesis, we did not delve into the correlation between human capabilities. Choosing which strategy to address a multi-label learning problem depends on the task domain. For example, in a kitchen scenario, the robot estimates two human capabilities: if the collaborator has the strength to open a refrigerator and if the collaborator has the strength to open a beer. There is likely a correlation between these two capabilities, making the Label Powerset more theoretically effective at estimating the probabilities of these two capabilities in this context. We have validated that the framework we proposed in this thesis can tackle such complex situations. Future research can further explore the correlation between human capabilities and the strategy selection for resolving the multi-label learning problem.

6.2.4 Continuous Task Domains in Reinforcement Learning

So far, the collaborative tasks we have experimented with have discrete state spaces and discrete action spaces. According to Table 6.1, we can find that both EST-Action and EST-State-Policy can partially support tasks in continuous state space or continuous action space. EST-Policy, on the other hand, imposes no specific limitations. Therefore, we expect to introduce collaborative studies with continuous state or action spaces. By doing this, we can further reveal the differences among the capability estimation strategies we presented.

6.2.5 Generalization Across Domains

In our existing experiments, we modeled each collaborative task and conducted pre-training for each capability combination. However, a refresh pre-training for all capability combinations is required when we wish to introduce a new capability to

an existing collaborative task. It would be interesting to research how to avoid pre-training from scratch. Furthermore, can the learning process for capability estimation be generalized to work in different domains without learning from scratch?

6.2.6 Diverse Evidence and Estimations

The feature vector included state and action information generated through Reinforcement Learning in our proposed estimation strategies. However, the domain of RL is rich with additional signals and information that could be used to enhance our feature space further.

Immediate Rewards: One promising direction for future work involves incorporating immediate rewards into the feature vector. Immediate rewards can carry valuable information about the consequences of actions taken in a specific state, and their inclusion can potentially improve the performance and robustness of the estimation.

Temporal Abstraction: RL traditionally operates at an action-level temporal scale. However, many real-world tasks involve higher-level decision-making and planning over longer time horizons. Future work could explore methods for abstracting and summarizing sequences of actions and states, potentially leading to more efficient and interpretable feature representations.

Multi-Agent Collaboration: Extending our approach to multi-agent environments is an exciting direction. Nursing homes and intelligent houses of the future may involve collaboration between multiple agents and different collaborators, and modeling their interactions can be challenging.

Furthermore, we view Reinforcement Learning as a method for capability estimation, not the only method. Future research can explore hybrid approaches combining RL with other machine learning or statistical methods to leverage their strengths and mitigate weaknesses.

6.3 Conclusion

This thesis presents a novel framework for online human capability estimation, applying Reinforcement Learning and Bayesian inference. We consider human capabilities as preconditions of robot actions. Our framework tracks evidence within the Reinforcement Learning context, offering three distinct strategies for capability estimation, each adapted to specific requirements. Additionally, we integrated pre-training policies and

introduced two capability-guided exploration strategies, further enhancing the agent's RL performance.

Through multiple collaborative experiments, including two simulator-based studies and the implementation of two popular Reinforcement Learning algorithms, Q-learning and DQN, we have demonstrated our framework's actual benefits in improving RL performance and accurate capability estimation. The real-world experiment has affirmed the feasibility of capability estimation, with clear results that agent performance can be substantially enhanced through capability-guided exploration.

Looking ahead, We expect to reduce pre-training requirements by refining the multi-label learning algorithm. Exploring the integration of diverse sources of evidence and advancing the generalization of capability estimation across a wider array of tasks represent promising directions for further research.

References

[1] Abdo, N., Stachniss, C., Spinello, L., and Burgard, W. (2015). Robot, organize my shelves! tidying up objects by predicting user preferences. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1557–1564.

[2] Al Qathrady, M. and Helmy, A. (2017). Improving ble distance estimation and classification using tx power and machine learning: A comparative analysis. In *Proceedings of the 20th ACM International Conference on Modelling, Analysis and Simulation of Wireless and Mobile Systems*, MSWiM '17, page 79–83, New York, NY, USA. Association for Computing Machinery.

[3] Aly, A. and Tapus, A. (2013). A model for synthesizing a combined verbal and nonverbal behavior based on personality traits in human-robot interaction. In *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 325–332.

[4] Arias-de Reyna, E. (2013). A cooperative localization algorithm for uwb indoor sensor networks. *Wireless personal communications*, 72(1):85–99.

[5] Aylett, R., Kappas, A., Castellano, G., Bull, S., Barendregt, W., Paiva, A., and Hall, L. (2015). I know how that feels — an empathic robot tutor. In *eChallenges e-2015 Conference*, pages 1–9.

[6] Boyan, J. and Moore, A. (1994). Generalization in reinforcement learning: Safely approximating the value function. In Tesauro, G., Touretzky, D., and Leen, T., editors, *Advances in Neural Information Processing Systems*, volume 7. MIT Press.

[7] Broz, F., Nourbakhsh, I., and Simmons, R. (2013). Planning for human–robot interaction in socially situated tasks: The impact of representing time and intention. *International journal of social robotics*, 5(2):193–214.

[8] Canal, G., Alenyà, G., and Torras, C. (2019). Adapting robot task planning to user preferences: an assistive shoe dressing example. *Autonomous robots*, 43(6):1343–1356.

[9] Canal, G., Torras, C., and Alenyà, G. (2021). Are preferences useful for better assistance? a physically assistive robotics user study. *J. Hum.-Robot Interact.*, 10(4).

[10] Canal Camprodon, G., Alenyà Ribas, G., and Torras, C. (2017). A taxonomy of preferences for physically assistive robots.

[11] Churamani, N., Anton, P., Brügger, M., Fließwasser, E., Hummel, T., Mayer, J., Mustafa, W., Ng, H. G., Nguyen, T. L. C., Nguyen, Q., Soll, M., Springenberg, S., Griffiths, S., Heinrich, S., Navarro-Guerrero, N., Strahl, E., Twiefel, J., Weber, C., and Wermter, S. (2017). The impact of personalisation on human-robot interaction in learning scenarios. In *Proceedings of the 5th International Conference on Human Agent Interaction*, HAI '17, page 171–180, New York, NY, USA. Association for Computing Machinery.

[12] Devin, S. and Alami, R. (2016). An implemented theory of mind to improve human-robot shared plans execution. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 319–326.

[13] Duque, I., Dautenhahn, K., Koay, K. L., Willcock, l., and Christianson, B. (2013). A different approach of using personas in human-robot interaction: Integrating personas as computational models to modify robot companions' behaviour. In *2013 IEEE RO-MAN*, pages 424–429.

[14] et al, C.-G. (2014). Learning behaviors by an autonomous social robot with motivations. *Cybernetics and systems*, 45(7):568–598.

[15] et al., M.-G. (2018). A bio-inspired motivational decision making system for social robots based on the perception of the user. *Sensors (Basel, Switzerland)*, 18(8):2691–.

[16] Fiore, M., Khambhaita, H., Milliez, G., and Alami, R. (2015). An adaptive and proactive human-aware robot guide. In *Social Robotics*, Lecture Notes in Computer Science, pages 194–203, Cham. Springer International Publishing.

[17] Gao, Y., Chang, H. J., and Demiris, Y. (2015). User modelling for personalised dressing assistance by humanoid robots. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1840–1845.

[18] Gibaja, E. and Ventura, S. (2015). A tutorial on multilabel learning. *ACM Computing Surveys (CSUR)*, 47(3):1–38.

[19] Godbole, S. and Sarawagi, S. (2004). Discriminative methods for multi-labeled classification. In *Advances in Knowledge Discovery and Data Mining*, Lecture Notes in Computer Science, pages 22–30, Berlin, Heidelberg. Springer Berlin Heidelberg.

[20] Görür, O. C., Rosman, B., Sivrikaya, F., and Albayrak, S. (2023). Fabric: A framework for the design and evaluation of collaborative robots with extended human adaptation. *J. Hum.-Robot Interact.* Just Accepted.

[21] Gross, H.-M., Boehme, H., Schroeter, C., Mueller, S., Koenig, A., Einhorn, E., Martin, C., Merten, M., and Bley, A. (2009). Toomas: Interactive shopping guide robots in everyday use - final implementation and experiences from long-term field trials. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2005–2012.

[22] Hasselt, H. (2010). Double q-learning. *Advances in neural information processing systems*, 23.

[23] Hasselt, H. v., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 2094–2100. AAAI Press.

[24] Hellou, M., Gasteiger, N., Lim, J. Y., Jang, M., and Ahn, H. S. (2021). Personalization and localization in human-robot interaction: A review of technical methods. *Robotics*, 10(4).

[25] Hemminghaus, J. and Kopp, S. (2017). Towards adaptive social behavior generation for assistive robots using reinforcement learning. In *2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI*, pages 332–340. ACM.

[26] Huang, C.-M. and Mutlu, B. (2016). Anticipatory robot control for efficient human-robot collaboration. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 83–90.

[27] Kanda, T., Shiomi, M., Miyashita, Z., Ishiguro, H., and Hagita, N. (2010). A communication robot in a shopping mall. *IEEE Transactions on Robotics*, 26(5):897–913.

[28] Karami, A. B., Sehaba, K., and Encelle, B. (2013). Adaptive and personalised robots - learning from users' feedback. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, pages 626–632.

[29] Karami, A. B., Sehaba, K., and Encelle, B. (2016). Adaptive artificial companions learning from users' feedback. *Adaptive behavior*, 24(2):69–86.

[30] Keene, D. R. (2015). A review of color blindness for microscopists: Guidelines and tools for accommodating and coping with color vision deficiency. *Microscopy and microanalysis*, 21(2):279–289.

[31] Klee, S. D., Ferreira, B. Q., Silva, R., Costeira, J. P., Melo, F. S., and Veloso, M. (2015). Personalized assistance for dressing users. In Tapus, A., André, E., Martin, J.-C., Ferland, F., and Ammi, M., editors, *Social Robotics*, pages 359–369, Cham. Springer International Publishing.

[32] Lam, C.-P. and Sastry, S. S. (2014). A pomdp framework for human-in-the-loop system. In *53rd IEEE Conference on Decision and Control*, pages 6031–6036.

[33] Malfaz, M. and Salichs, M. (2009). Learning to deal with objects. In *2009 IEEE 8th International Conference on Development and Learning*, pages 1–6. IEEE.

[34] Martins, G. S., Ferreira, P., Santos, L., and Dias, J. (2016). A context-aware adaptability model for service robots. In *IJCAI-2016 Workshop on Autonomous Mobile Service Robots*.

[35] Martins, G. S., Santos, L., and Dias, J. (2019a). Bum: Bayesian user model for distributed learning of user characteristics from heterogeneous information. *IEEE transactions on cognitive and developmental systems*, 11(3):425–434.

[36] Martins, G. S., Santos, L., and Dias, J. (2019b). User-adaptive interaction in social robots: A survey focusing on non-physical interaction. *International journal of social robotics*, 11(1):185–205.

[37] Matsubara, T., Miro, J. V., Tanaka, D., Poon, J., and Sugimoto, K. (2015). Sequential intention estimation of a mobility aid user for intelligent navigational assistance. In *2015 24th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 444–449. IEEE.

[38] McColl, D. and Nejat, G. (2013). Meal-time with a socially assistive robot and older adults at a long-term care facility. *J. Hum.-Robot Interact.*, 2(1):152–171.

[39] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning.

[40] Müller, S., Sprenger, S., and Gross, H.-M. (2014). Online adaptation of dialog strategies based on probabilistic planning. In *The 23rd IEEE International Symposium on Robot and Human Interactive Communication*, pages 692–697.

[41] Nikolaidis, S., Kuznetsov, A., Hsu, D., and Srinivasa, S. (2016). Formalizing human-robot mutual adaptation: A bounded memory model. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 75–82.

[42] Norcio, A. and Stanley, J. (1989). Adaptive human-computer interfaces: a literature survey and perspective. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(2):399–408.

[43] Park, H. W., Grover, I., Spaulding, S., Gomez, L., and Breazeal, C. (2019). A model-free affective reinforcement learning approach to personalization of an autonomous social robot companion for early literacy education. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press.

[44] Portugal, D., Santos, L., Alvito, P., Dias, J., Samaras, G., and Christodoulou, E. (2015). Socialrobot: An interactive mobile robot for elderly home care. In *2015 IEEE/SICE International Symposium on System Integration (SII)*, pages 811–816.

[45] Qureshi, A. H., Nakamura, Y., Yoshikawa, Y., and Ishiguro, H. (2018). Intrinsically motivated reinforcement learning for human–robot interaction in the real-world. *Neural networks*, 107:23–33.

[46] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8.

[47] Ray, C., Mondada, F., and Siegwart, R. (2008). What do people expect from robots? In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3816–3821. IEEE.

[48] Ros, R., Baroni, I., and Demiris, Y. (2014). Adaptive human–robot interaction in sensorimotor task instruction: From human to robot dance tutors. *Robotics and Autonomous Systems*, 62(6):707–720.

[49] Sarabia, M., Lee, K., and Demiris, Y. (2015). Towards a synchronised grammars framework for adaptive musical human-robot collaboration. In *2015 24th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 715–721.

[50] Schapire, R. E. and Singer, Y. (2000). Boostexter: A boosting-based system for text categorization. *Machine learning*, 39(2):135–168.

[51] Sekmen, A. and Challa, P. (2013). Assessment of adaptive human–robot interactions. *Knowledge-Based Systems*, 42:49–59.

[52] Senft, E., Baxter, P., Kennedy, J., Lemaignan, S., and Belpaeme, T. (2017). Supervised autonomy for online learning in human-robot interaction. *Pattern recognition letters*, 99:77–86.

[53] Sheridan, T. B. (2020). A review of recent research in social robotics. *Current Opinion in Psychology*, 36:7–12. Cyberpsychology.

[54] Taha, T., Miro, J. V., and Dissanayake, G. (2011). A pomdp framework for modelling human interaction with assistive robots. In *2011 IEEE International Conference on Robotics and Automation*, pages 544–549. IEEE.

[55] Tapus, A., Ţăpuş, C., and Matarić, M. J. (2008). User—robot personality matching and assistive robot behavior adaptation for post-stroke rehabilitation therapy. *Intelligent service robotics*, 1(2):169–183.

[56] Tseng, S.-H., Liu, F.-C., and Fu, L.-C. (2018). Active learning on service providing model: Adjustment of robot behaviors through human feedback. *IEEE transactions on cognitive and developmental systems*, 10(3):701–711.

[57] Tsiakas, K., Abujelala, M., and Makedon, F. (2018). Task engagement as personalization feedback for socially-assistive robots and cognitive training. *Technologies (Basel)*, 6(2):49–.

[58] Tsoumakas, G. and Katakis, I. (2007). Multi-label classification: An overview. *International journal of data warehousing and mining*, 3(3):1–13.

[59] Tsoumakas, G., Katakis, I., and Vlahavas, I. (2010). *Mining Multi-label Data*, pages 667–685. Springer US, Boston, MA.

[60] Tsoumakas, G. and Vlahavas, I. (2007). Random k-labelsets: An ensemble method for multilabel classification. In Kok, J. N., Koronacki, J., Mantaras, R. L. d., Matwin, S., Mladenič, D., and Skowron, A., editors, *Machine Learning: ECML 2007*, pages 406–417, Berlin, Heidelberg. Springer Berlin Heidelberg.

[61] Wang, Y., Yang, X., Zhao, Y., Liu, Y., and Cuthbert, L. (2013). Bluetooth positioning using rssi and triangulation methods. In *2013 IEEE 10th Consumer Communications and Networking Conference (CCNC)*, pages 837–842. IEEE.

[62] Wang, Z., Boularias, A., Mülling, K., Schölkopf, B., and Peters, J. (2017). Anticipatory action selection for human–robot table tennis. *Artificial Intelligence*, 247:399–414. Special Issue on AI and Robotics.

[63] Yedidsion, H., Suriadinata, J., Xu, Z., Debruyn, S., and Stone, P. (2021). A scavenger hunt for service robots.

[64] Zhang, M.-L. and Zhang, K. (2010). Multi-label learning by exploiting label dependency. In *Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 999–1008. ACM.

[65] Zhang, M.-L. and Zhou, Z.-H. (2014). A review on multi-label learning algorithms. *IEEE transactions on knowledge and data engineering*, 26(8):1819–1837.