
Learning to Solve Complex Control Tasks via Hierarchical Strategies

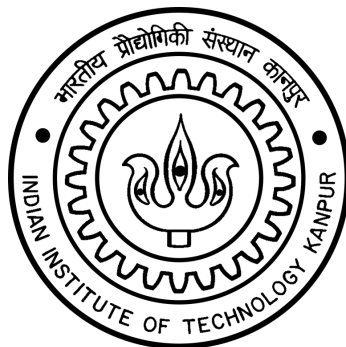
*A thesis submitted in partial fulfilment of the requirements for the award of the
degree of*

Doctor of Philosophy

by

Utsav Singh

(Roll number: 16511261)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR**

March 2025



Certificate

It is certified that the work contained in this thesis entitled “**Learning to Solve Complex Control Tasks via Hierarchical Strategies**” by **Utsav Singh** has been carried out under our supervision and that it has not been submitted elsewhere for a degree.

A handwritten signature in black ink, appearing to read "Sunil".

Dr. Sunil E. Simon
CSE Department
Indian Institute of Technology Kanpur

A handwritten signature in black ink, appearing to read "Vinay Namboodiri".

Dr. Vinay P. Namboodiri
Department of Computer Science
University of Bath

March 2025

Declaration

This is to certify that the thesis titled “**Learning to Solve Complex Control Tasks via Hierarchical Strategies**” has been authored by me. It presents the research conducted by me under the supervision of **Dr. Sunil E. Simon** and **Dr. Vinay P. Namboodiri**.

To the best of my knowledge, it is an original work, both in terms of research content and narrative, and has not been submitted elsewhere, in part or in full, for a degree. Further, due credit has been attributed to the relevant state-of-the-art and collaborations with appropriate citations and acknowledgments, in line with established norms and practices.

Utsav Singh

Roll No. 16511261

CSE Department

Indian Institute of Technology Kanpur

Abstract

Name of the student: **Utsav Singh**

Roll No: **16511261**

Degree: **Doctor of Philosophy** Department: **Computer Science and Engineering**

Thesis title: **Learning to Solve Complex Control Tasks via Hierarchical Strategies**

Thesis supervisors: **Dr. Sunil E Simon and Dr. Vinay P Namboodiri**

Month and year of thesis submission: **March 2025**

Humans have long been fascinated with artificial intelligence (AI), which has led to ground-breaking discoveries such as solving the game of Go [1], solving complex reasoning tasks using large language models [2] and predicting protein structures with alphafold [3]. These advancements have also been adapted to steer progress in solving robotic control tasks, including robotic navigation and manipulation [4]. However, it has been really challenging for AI agents to solve complex long horizon tasks which involve long term decision making, due to issues like inefficient exploration and long term credit assignment, especially in sparse reward scenarios.

For instance, a robotics agent trained on “opening a microwave door” will fail miserably with multi-stage tasks like “opening the microwave door, taking a bowl out, placing it on a stove, and turning on the gas knob”. Humans on the other hand, are naturally efficient at solving such tasks by dividing them into simpler sub-tasks, and then composing their solutions to solve the overall objective. This hierarchical problem-solving ability is supported by evidence from cognitive psychology and is strongly linked to the functions of the prefrontal cortex, which is responsible for goal representation, task switching, and hierarchical control[5].

Inspired by this, we posit that hierarchical abstraction is key to enabling robotic agents to solve complex sparsely rewarded tasks. To this end, we propose efficient hierarchical strategies to enable robotic agents to break long-horizon tasks into manageable subtasks, solve these subtasks, and compose their solutions effectively to achieve the overall objective.

We perform extensive experiments across diverse robotic navigation and manipulation tasks and demonstrate that these hierarchical approaches are able to consistently outperform existing baselines, and show impressive performance in challenging environments. Extensive experiments on real world robots validate the practicality and robustness of the proposed methods. Thus, in this dissertation, we have put forward some initial efforts towards a larger goal of building practical robotic systems that can autonomously solve complex real-world tasks.

To my parents - Rekha and Narendra Singh
and my brother - Kartik Singh

Acknowledgements

If I have seen further, it is by standing on the shoulders of giants.

– Isaac Newton

When I recall the beginning of my PhD journey, I have faint memories of myself as a confused masters student struggling to find his calling. Although I had no idea of the adversities of the PhD journey ahead, my only aim was to perform good research, which ultimately contributes to the betterment of the society and life in general. This has been my guiding motivation throughout this journey that has kept me focused and sincere throughout. However, although I happen to get the credit for this work, this work would not have been even remotely possible without the collaborative efforts, guidance, and support of the Almighty, my supervisors, teachers, parents, brother, collaborators, friends and most of all, the many many failures along the way. I am really grateful to all of them, for they have molded me from a faint memory into the person I am today.

I would first like to thank the God and Guruji for constantly taking care of me, and blessing me with strength and capability throughout the process. It is said that true prayer happens either in gratefulness or in deep misery. Well I can say for a fact that my PhD journey has given me many many reasons to pray!

This dissertation would not have been possible without the unconditional love and support of my parents. Ever since I was young, both my parents motivated me to ask relevant questions, go deep into a discussion and silently comprehend and analyze the problem. I still remember how before any exams, my parents would shield me from any source of distractions. As I sat studying, my father would sit beside me everyday, while studying his own books. I am grateful for my father and mother's unconditional love, and they have

instilled in me the values of honesty and hard work, while maintaining a balance between research, mental health and meditation, fitness and music. I would not have been here without the unwavering support of my brother, who has always been a constant source of companionship and deep critical intellectual discussions. I am also very grateful for the support of my aunt Brijesh, cousins Pratima and Nishant, and uncle Ramesh and the extended members of my family for always supporting me.

I will forever be grateful to IIT Kanpur institute, the IITK CSE department and the professors for teaching and explaining to me the basics of computer science. I would especially like to thank my supervisors Prof Vinay P Namboodiri and Prof Sunil E Simon for guiding me throughout my journey, accommodating all my requests and mistakes, and constantly acting as an anchor to my PhD ship. I had many fruitful discussions with Prof. Vinay throughout my PhD, which would sometimes go deep into a relevant problem, and sometimes even turn philosophical. Prof Vinay has been more than a mentor to me, who introduced me to the area of reinforcement learning, deeply impacted my knowledge of computer vision, and provided me the resources, environment and freedom to do research, for which I will forever be grateful. Prof Sunil has been a constant support to me especially towards the end of my PhD, and I will always cherish our discussions on important problems on hierarchical learning and game theory. This research was partially funded by the ANRF under the grant CRG/2022/006140.

I would also like to thank Dr. Pulkit Agarwal for guiding me through an internship I did at Bangalore and during my initial PhD. I have hardly seen the level of clarity related to ML research elsewhere, and I would thoroughly enjoy our short fruitful discussions whenever I got the chance. I would also like to thank Anurag Ajay for guiding and training me at my first PhD project. I learnt a lot from him, especially about how to approach a problem, and how to present the research work in a comprehensible manner for the reader.

I would like to thank Dr. Amrit Singh Bedi for his constant guidance during the second half of my PhD work. It was from him that I learnt how to pick a research problem, the importance of being aware of current research, and to clearly think about an approach

theoretically and practically. Prof Amrit, Wesley Suttle and Souradip Chakraborty taught me how to properly formulate the story of a research paper, go deep into the root problem, and how to properly write and present good research papers.

I was lucky enough to find great friends at IIT Kanpur like Poonam, Lavish, Aditi, Nimisha, Priya, Rahul Vishwakarma, Rahul Bhardwaj, Saif, KP, Puneet, Mahesh, Keddia, Garima, and for my friends from my bachelors Vipul, Utkarsh, Amit, Ashutosh, and Vinay for their constant support throughout my PhD journey. I still remember I would always remember Samik whenever there was a systems issue and I knew I could call him even at 4 am at night. I will remember the evening tea sessions at CC canteen where I would nudge everyone to eat healthy but eventually had to cave in. My PhD at IIT Kanpur would have definitely been a lot less fun without these people. I would also like to thank the CSE office for aiding my office work, and to Hall 11 for being an amazing home away from home during my PhD.

My fondest memories of PhD lie with the Adventure Club of IIT Kanpur and all its amazing members, for it was here that I was introduced to the world of high altitude trekking, adventure sports, marathons and fitness activities. In their companionship, I was fortunate enough to attempt and complete 14 treks and cycling expeditions, many marathons, the 72km Khardungla challenge and the Ironman 70.3 during my stay at IIT Kanpur.

I would also like to thank my vocal music teachers Pandit Umakant Pathak and Sweta Bharti for teaching me the intricacies of indian classical music, and for their patient and dedicated vocal training.

Contents

List of Figures	xiii
Abbreviations	xv
1 Introduction	1
1.1 Problem Formulation	2
1.2 HRL: Hierarchical Reinforcement Learning	4
1.2.1 Mathematical Formulation of the HRL Framework	8
1.3 Illustrative Example: Robotic Tea-Making.	10
1.4 HRL: Design Challenges	13
1.5 Proposed Solution	15
1.5.1 Developing evaluation benchmarks	15
1.5.2 Leveraging expert demonstrations	15
1.5.3 Leveraging preference-based learning	16
1.5.4 Leveraging natural language instructions	16
2 Related Work	18
2.1 Advancements in Dynamic Programming, Artificial Intelligence, and Machine Learning	18
2.1.1 Dynamic Programming.	18
2.1.2 Artificial Intelligence.	19
2.1.3 Machine Learning and Reinforcement Learning.	19
2.2 Foundations of Hierarchical Control and Temporal Abstraction	20
2.2.1 Early Hierarchical RL in Practice	21
2.2.2 Hierarchical RL and Planning in Decision Making and Robotics	21
2.2.3 Hierarchical RL in Cognitive Science and Neuroscience	22
2.2.4 Automatic Discovery of Temporal Abstractions and Options	22
2.3 Recent Advances: Deep HRL and Large-Scale Autonomous Agents	22
2.4 Summary	23
2.5 Open Problems and Future Directions	23
3 Problem Definitions	24
3.1 Markov Decision Processes (MDP)	24

3.2	Goal-Conditioned Reinforcement Learning	25
3.3	Hierarchical Reinforcement Learning (HRL)	26
3.3.1	Hierarchical Policy Decomposition	26
3.3.2	Execution Protocol	26
3.3.3	Reward Structure and Buffers	27
3.3.4	Key Challenges in HRL	27
3.4	Preference-Based Reinforcement Learning	28
3.4.1	Bradley-Terry Model	28
3.4.2	RLHF: Reinforcement Learning from Human Feedback	28
3.4.3	Direct Preference Optimization (DPO)	29
3.4.3.1	Limitations of directly applying PbL to HRL	29
3.5	Language to Rewards (L2R)	30
3.5.1	Limitations of L2R framework	30
3.6	Problem Statement	30
I	Designing Evaluation Frameworks for Complex Control Tasks	33
4	Robust Evaluation Frameworks for Complex Control Tasks	34
4.1	Environments and their implementation details	36
4.1.1	Randomized maze navigation task	36
4.1.2	Pick and place, push, bin and hollow environments	37
4.1.3	Rope Manipulation Environment	39
4.1.4	Franka kitchen Environment	39
4.2	Generating expert demonstrations	40
4.2.1	Maze navigation Environment	40
4.2.2	Pick and place and Push Environment	40
4.2.3	Bin Environment	41
4.2.4	Hollow Environment	41
4.2.5	Rope Manipulation Environment	41
4.2.6	Kitchen Environment	42
4.3	Real World Experiments	42
II	Leveraging Expert Demonstrations	44
5	CRISP: Curriculum-inducing Primitive Informed Subgoal Prediction	45
5.1	Curriculum-Inducing Subgoal Prediction	48
5.1.1	Primitive Informed Parsing: PIP	49
5.1.2	Inverse RL regularization	51
5.1.3	Joint optimization	52
5.2	Experiments	53
5.2.1	Additional hyper-parameters	53
5.2.2	Environment and Implementation details	55
5.2.3	RAPS baseline setup	55

5.3	Related Work	66
5.4	Discussion	67
6	PEAR: Primitive Enabled Adaptive Relabeling for Boosting Hierarchical Reinforcement Learning	68
6.1	Primitive Enabled Adaptive Relabeling	70
6.1.1	PEAR	71
6.1.2	Adaptive Relabeling	72
6.1.3	Periodic re-population of higher level subgoal dataset	72
6.1.4	Dealing with out-of-distribution states	74
6.1.5	Joint optimization	74
6.1.6	Sub-optimality analysis	76
6.1.7	Proofs for Theorem 1 for higher and lower level policies	78
6.1.7.1	Sub-optimality proof for higher level policy	78
6.1.7.2	Sub-optimality proof for lower level policy	79
6.2	Experiments	82
6.2.1	Evaluation and Results	83
6.2.2	Qualitative Insights	86
6.2.3	Ablative analysis	87
6.2.3.1	Dealing with non-stationarity and infeasible subgoal generation in HRL	88
6.2.3.2	Additional Ablations	89
6.2.3.3	Real world experiments	90
6.3	Related Work	92
6.4	Discussion	95
III	Aligning Control with Preferences	97
7	PIPER: Primitive-Informed Preference-based Hierarchical Reinforcement Learning via Hindsight Relabeling	98
7.0.1	Limitations of Existing Approaches to HRL	100
7.1	Primitive Informed Preference Based Learning	101
7.1.1	Preference-based Learning (PBL)	102
7.1.2	PiL: Primitive-in-the-Loop Feedback for HRL	104
7.1.3	PiL with Goal-conditioned Hindsight Relabeling	106
7.1.4	Primitive-informed Regularization	107
7.1.4.1	Deriving the final optimum of KL-Constrained Reward Maximization Objective	108
7.1.5	PIPER Implementation	111
7.2	Experiments	111
7.2.1	Setup	112
7.2.1.1	Additional hyper-parameters	114
7.2.2	Evaluation and Results	114
7.2.3	Qualitative Insights	117

7.2.4	Ablation Analysis	118
7.3	Related Work	120
7.4	Discussion	123
8	Hierarchical Preference Optimization: Learning to Achieve Goals via Feasible Subgoals Prediction	125
8.1	HPO : Hierarchical Preference Optimization	132
8.1.1	HRL: Bi-Level Formulation	133
8.1.2	HPO	135
8.1.3	HPO Algorithm	138
8.2	Experiments	138
8.3	Related Work	145
8.4	Discussion	146
IV	From Words to Actions	148
9	LGR2: Language Guided Reward Relabeling for Accelerating Hierarchical Reinforcement Learning	149
9.1	Language Guided Reward Relabeling	152
9.1.1	Hierarchical Setup.	152
9.1.2	Language to Rewards.	153
9.1.3	Challenges of Language-to-Reward (L2R) Approaches.	154
9.1.4	Reward Parameter Generation	154
9.1.5	Relabeling of High-Level Rewards	156
9.1.6	Addressing Reward Sparsity with HER	156
9.1.7	Training Objectives	157
9.1.7.1	Lower-Level Policy Loss	157
9.1.7.2	Higher-Level Policy Loss	158
9.2	Experiments	159
9.3	Full Prompts	167
9.3.1	Qualitative Behavioral Explanations	176
9.4	Sample Codes	176
9.5	Related Work	180
9.6	Discussion	182
10	Conclusion	184
10.1	Specific Contributions to Hierarchical Reinforcement Learning	185
10.2	Comparative Summary of Method Contributions:	185
10.3	Positioning Within Current HRL Research	186
10.4	Common Failure Cases and Mitigation Strategies	187
10.5	Broader Impact and Future Directions Within HRL	188
	References	190

List of Figures

1.1	Higher-level Neural Network Architecture. The network takes state and final goal as inputs and processes them through hidden layers to output subgoals.	10
1.2	Lower-level Neural Network Architecture. The network takes state and subgoal as inputs and processes them through hidden layers to output primitive actions.	10
1.3	HRL: Illustrative Example	11
3.1	HRL: Execution Protocol	26
4.1	Maze navigation task visualization	37
4.2	Pick and place task visualization	38
4.3	Push task visualization	38
4.4	Bin task visualization	38
4.5	Hollow task visualization	38
4.6	Rope manipulation task visualization	39
4.7	Franka kitchen task visualization	40
4.8	Real world tasks visualization	43
5.1	CRISP: PIP Overview	47
5.2	CRISP: Subgoal curriculum generated during training	48
5.3	CRISP: Ablation experiment: CRISP vs CRISP-RPL ablation	57
5.4	CRISP: Success rate comparison against baselines	58
5.5	CRISP: Ablation experiment: CRISP vs CRISP-RPL ablation	59
5.6	CRISP: Non-stationarity metric comparison	61
5.7	CRISP: Ablation experiment: Learning rate ablation	64
5.8	CRISP: Ablation experiment: Population hyper-parameter ablation	65
5.9	CRISP: Ablation experiment: Number of expert demonstrations ablation	65
5.10	CRISP: Ablation experiment: RPL window size ablation	66
6.1	PEAR: Adaptive Relabeling Overview	69
6.2	PEAR: Subgoal Evolution	70
6.3	PEAR: Success rate comparison	84
6.4	PEAR: Non-stationarity metric comparison	88
6.5	PEAR: Feasible subgoal generation metric comparison	89
6.6	PEAR: PEAR vs PEAR-RPL Ablation	90

6.7	PEAR: PEAR vs PEAR-No-Margin Ablation	91
6.8	PEAR: Q_{thresh} hyper-parameter ablation	91
6.9	PEAR: Population hyper-parameter ablation	92
6.10	PEAR: RPL Window Ablation	92
6.11	PEAR: Learning Rate Ablation	93
6.12	PEAR: Number of expert demonstrations Ablation	93
6.13	PEAR: PEAR vs HER-BC Ablation	94
6.14	PEAR: PEAR vs HBC vs HBC-demos Ablation	95
6.15	PEAR: Sub-optimal Demonstartions Ablation	96
7.1	PIPER Overview	99
7.2	PIPER: Success Rate Comparison	115
7.3	PIPER: Hindsight Relabeling Ablation	119
7.4	PIPER: Hindsight Relabeling Ablation	120
7.5	PIPER: Target networks Ablation	121
7.6	PIPER: Dense Rewards Ablation	122
8.1	HPO Overview	126
8.2	HPO: Success Rate Comparison	139
8.3	HPO: Subgoal Distance Metric	142
8.4	HPO: Lower Q-Function Metric	142
8.5	HPO: Regularization Weight Ablation	145
8.6	HPO: Max-ent Parameter Ablation	145
9.1	LGR2 Overview	150
9.2	LGR2: Success Rate Comparison	159
9.3	LGR2: Comparison against HRL+HER	161
9.4	LGR2: Non-Stationarity Metric	161
9.5	LGR2: Reward Comparison	162
9.6	LGR2: Hindsight Relabeling Ablation	163

Abbreviations

AI	A rtificial I ntelligence
ML	M achine L earning
RL	R einforcement L earning
HRL	H ierarchical R einforcement L earning
LfD	L earning from D emonstrations
IL	I mitation L earning
BC	B ehavior C loning
IRL	I nverse R einforcement L earning
PbL	P reference b ased L earning
LLM	L arge L anguage M odel
CRISP	C urriculum-inducing P rimitive I nformed S ubgoal P rediction
PEAR	P rimitive E nabled A daptive R elabeling
PIPER	P rimitive-Informed P reference-based HRL via Hindsight R elabeling
HPO	H ierarchical P reference O ptimization
LGR2	L anguage G uided R eward R elabeling
HAC	H ierarchical A ctor C ritic

Chapter 1

Introduction

Divide each difficulty into as many parts as is feasible and necessary to resolve it.

– René Descartes

Humans are naturally efficient at solving complex multi-stage tasks by dividing them into smaller sub-tasks, solving them and composing their solutions to solve the overall long horizon task. For instance, humans are able to easily solve tasks like making breakfast, which involve solving many sub-tasks sequentially, for example: putting bread in the toaster, applying butter on the toasted bread, pouring juice in a glass, etc. Machines on the other hand, find it extremely difficult to solve such multi-stage tasks. For example, a robot trained to pick up laundry will fail miserably if assigned the complex multi-stage task of opening the washing machine, picking up the laundry, and then folding the laundry on the table. Prior work suggests that this challenge arises due to two key issues: inefficient exploration and ineffective long-term credit assignment [6]. Concretely, in long-horizon tasks, the robot may struggle to explore the environment effectively or fail to accurately assign credit to the specific actions responsible for accurately solving the task.

1.1 Problem Formulation

The primary aim of artificial intelligence is to devise general-purpose autonomous agents that are able to solve complex tasks in real-world environments. One influential perspective frames intelligent agents through an *economic* lens [7]: the agent dynamically interacts with its environment and receives real-valued feedback in the form of rewards and penalties, with the overarching objective of maximizing its cumulative reward. This perspective offers a rich framework to express and solve a myriad of real-world problems, where the positive rewards can be associated with successful outcomes, whereas negative rewards can penalize undesirable events. This framework also facilitates reasoning under uncertainty, requiring agents to optimize their expected return amidst stochastic, unpredictable events in the environment [8].

However, this powerful framework leads to the significant research challenge of devising planning and learning algorithms that scale efficiently in high-dimensional state spaces. Early work in stochastic planning and dynamic programming demonstrated optimality given complete probabilistic models [9], but suffered from prohibitive computational costs in large domains. Dynamic programming methods from early planning and control theoretically solve hard problems optimally, but their cost scales polynomially with state space size, rendering them impractical for high-dimensional or long-horizon robotic tasks [10]. Consequently, recent research has shifted toward scalable approaches that exploit underlying task structure, culminating in the rise of reward-based and reinforcement learning paradigms [11] as practical and general-purpose solutions.

Reward-based learning formalizes intelligent decision-making as a process in which an agent seeks to maximize cumulative numerical rewards while adapting to environmental feedback. Moreover, the uncertainty and stochasticity aspects are handled by focusing on maximizing the expected cumulative reward across random outcomes. This approach draws inspiration from operant conditioning in psychology and underpins most modern reinforcement learning systems [11]. By combining online learning from direct experience

and the use of approximation for representing complex value functions or policies, agents learn effective strategies that generalize even in vast or continuous state spaces.

Reinforcement learning (RL) [12] has recently enabled breakthroughs such as Deep Q-Networks (DQN) [13] which outperforms humans at playing games, AlphaGo [1] which plays the complex game of Go at super-human level, predicting protein structures with alphafold [3], etc. Such breakthroughs have showcased the ability of RL to solve complex tasks. Such advancements have been leveraged to also steer progress in robotic control tasks, including navigation and manipulation [4]. However, most of these advancements are on short-horizon tasks, and seemingly straightforward multi-stage tasks can be challenging for robots due to limitations in perception, manipulation, adaptation, and coordination.

Despite major progress in both probabilistic planning and reinforcement learning (RL), most widely used algorithms remain fundamentally “flat”, treating the state space as a monolithic entity without abstraction [9, 11]. In RL and planning, the cost of propagating information about future reward along long flat trajectories remains a daunting bottleneck. Concretely, complexity in control tasks grows rapidly with task horizon, branching factor, and the richness of state spaces. Classic RL methods, while effective for short-horizon settings, often become ill-suited for such problems.

In contrast, humans routinely solve long-horizon tasks that remain exceptionally challenging for artificial agents. This naturally raises the question: *What enables humans to efficiently solve such complex, multi-step problems?*

Hierarchical Learning

The early literature in artificial intelligence, planning, and cognitive science observed that both animal and human intelligence rely on hierarchical organization: abstract, temporally extended plans decompose into sequences of subtasks and, eventually, low-level actions [14, 15]. Classical planning research has shown that hierarchical decompositions, such as hierarchical task networks (HTNs) [16], macro actions [17, 18], and state abstraction [19, 20] can provide exponential reductions in computational cost.

In this thesis, we take inspiration from human evolution (a process that has spanned approximately 18 million years) and posit that hierarchical learning in the human brain is deeply connected to the way we structure, process, and organize information to efficiently solve complex problems and make decisions. Advancements in cognitive psychology and neuroscience have shown that the brain is able to break down complex tasks into smaller, more manageable sub-tasks and learn hierarchical structures from experience, enabling efficient decision-making, problem-solving, and adaptation [14]. The human prefrontal cortex (PFC) is a brain region central to complex cognitive functions like decision-making, problem-solving, and goal-directed behavior. Research in neuroscience has shown compelling evidence that the PFC supports hierarchical learning and problem-solving, which means that it can manage information at multiple levels of abstraction [15].

Hierarchical learning leverages temporal abstraction and task decomposition, aiming to mirror the way humans decompose complex problems into manageable subtasks [14, 15]. The core motivation is that breaking down a problem along meaningful sub-tasks that allows agents to focus learning and planning where it is most needed, while facilitating generalization and skill reuse. In this thesis, we develop mathematical models for hierarchical learning by taking inspiration from human cognitive psychology and neuroscience.

We now formally specify the hierarchical reinforcement learning (HRL) framework pursued in this thesis, and rigorously define the HRL architecture and its constituent policy, state, and reward structures, reflecting the key settings of our task domains. We then enumerate the central design challenges in hierarchical RL, including issues like non-stationarity. Subsequently, we enlist our proposed solutions to address these issues to tackle the core theoretical and practical challenges inherent to hierarchical RL.

1.2 HRL: Hierarchical Reinforcement Learning

Hierarchical reinforcement learning (HRL) extends the standard reinforcement learning framework by explicitly introducing multiple levels of temporal and structural abstraction.

This enables an agent to decompose complex, long-horizon tasks into tractable sub-tasks, each associated with its own decision-making policy, reward structure, and planning horizon. Such decomposition mirrors the way humans solve complex problems by organizing behavior into hierarchies of skills, strategies, and subtasks [14, 15]. As discussed before, the early inspirations for HRL can be found in the neuroscience of prefrontal cortex organization, AI planning, and robotics, while practical instantiations range from options [21], MAXQ [22], and feudal learning [23] to more recent deep-HRL methods [24]. In this thesis, we focus on feudal architecture to devise HRL algorithms. We consider a two-level goal-conditioned HRL setup, though the principles described extend to deeper or more structured hierarchies as well.

Design Principles.

Before establishing the mathematical formulation of HRL, let us first understand the design principles behind the hierarchical framework [25]. This is essential for grasping the design challenges inherent in HRL and will offer valuable insights into potential solutions.

1. Decompositional Abstraction for Complex Domains

Motivation: Directly learning policy mappings in complex domains often proves intractable, particularly those characterised by delayed rewards, partial observability, or long time horizons. To address this, such tasks can be decomposed into a hierarchy of simpler sub-tasks, making the learning process more manageable and effective.

Application: In our hierarchical framework, the overall task is structured into multiple levels, such as a high-level manager and a low-level worker, with each level operating at its own temporal scale. The high-level manager is responsible for generating subgoals, which the low-level worker strives to achieve through its learned policy. This hierarchical structure is broadly applicable to any task matching these criteria.

While various hierarchical abstractions fall within the broader scope of our decompositional approach, for all simulation and real-world experiments discussed in this thesis, we focus on the following specific hierarchical abstraction:

Task Parameter	Value
State/Action	Real-valued, continuous
Dimensionality	Moderate-high ($D > 10$)
Hierarchy Depth	Two levels
Subgoal Representation	Spatial/object-centric waypoints/state space
Agent Structure	Single 7-DoF/9-DoF robotic arm
Reward	Sparse, goal-driven

TABLE 1.1: Task parameters for the hierarchical methods proposed in this thesis.

2. Domain-Driven Task Decomposition and Granularity

Motivation: The decomposition of complex tasks into constituent skills and behaviors should be guided by the semantics and demands of the specific domain. Depending on the characteristics of the domain, hierarchical abstractions can be implemented using fixed, domain-specific templates or by leveraging data-driven approaches where certain aspects of the hierarchy are learned. For instance, the task of making tea can be naturally decomposed into sequential subtasks such as pouring tea and water into a kettle and then placing the kettle on the stove. The identification of such sub-tasks can either rely on expert domain knowledge or be discovered autonomously by learning algorithms.

Application: In our framework, the higher-level policy predicts subgoals, which the lower-level policy is trained to achieve. The nature of these subgoals is domain-dependent: for navigation tasks, the subgoal may be a spatial waypoint; for robotic manipulation tasks, it could be a specific robot configuration (e.g., end-effector position and joint torques); and for kitchen tasks, the subgoal might be the current state of the environment. Thus, we adapt the choice of subgoal representation to best suit the demands of each particular task.

Notably, our hierarchical setup is designed to require minimal domain-specific task decomposition effort from experts.

3. Learning Skill Acquisition

Motivation: Machine learning empowers agents to autonomously acquire, refine, and adapt the necessary skills at each hierarchical level. Rather than relying solely on manual design, the skills required at different layers are learned using various machine learning algorithms, such as supervised learning with labeled data or reinforcement learning from interaction feedback. This flexibility allows the hierarchical framework to accommodate any suitable learning based approach.

In some instances, lower-level skills or behaviours may be predetermined by either being explicitly programmed by domain experts or learned in advance via machine learning techniques. Once established, these skills are used as fixed modules when training or deploying higher-level behaviours. Hierarchical architectures thus offer the flexibility to combine learned and hand-crafted skills as appropriate in various domains, supporting a range of approaches for skill acquisition and composition.

Application:

- Use offline learning for skills whose specifications do not change and can be honed with curated datasets.
- Employ online learning to adapt both high and low-level behaviours in response to domain-specific environmental dynamics, stochasticity, and new objectives.
- Select learning algorithms (e.g., neural networks, policy gradients, Q-learning, imitation learning, preference-based learning) that fit the demands of each subtask and abstraction level.

4. Inter-level Interaction and Cooperation

Motivation: The essence of hierarchical learning is that progress or changes in one layer concretely influence learning efficiency and representation in adjacent layers. For example, in feudal hierarchical framework, after the higher level predicts a subgoal, its next achieved state depends on the lower level behavior. Thus, various hierarchical levels interact and cooperate together to solve the complete task.

Application: Subtasks solved by lower layers provide foundation for higher-level strategies (e.g., a skill like “pick up object” forms the core of “arrange items on table”). Crucially, the learned representations, policies, or rewards at one level define the effective action space or state input to the next higher level. A good subgoal prediction should ideally balance the task-split between hierarchical levels to prevent any degenerate solutions.

1.2.1 Mathematical Formulation of the HRL Framework

Base MDP Setup.

The reinforcement learning problem is customarily framed as a Markov Decision Process (MDP), specified by the tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where:

- $\mathcal{S}$ is the state space,
- $\mathcal{A}$ is the action space,
- $p : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$ the transition function,
- $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ the reward function,
- $\gamma \in (0, 1)$ the discount factor.

The agent seeks to optimize a policy $\pi : \mathcal{S} \rightarrow \Delta(\mathcal{A})$ so as to maximize the expected return

$$J(\pi) = \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]. \quad (1.1)$$

We explain the higher and lower level MDPs as follows:

Higher-Level MDP: The higher-level MDP is defined as $(\mathcal{S}, \mathcal{G}, \mathcal{A}^H, p^H, r^H, \gamma^H)$, where $\mathcal{S}$ is the state space, $\mathcal{G}$ is a (sub)goal space, and the higher-level policy $\pi^H : \mathcal{S} \rightarrow \Delta(\mathcal{A}^H)$ proposes subgoals or options $g_t \sim \pi^H(\cdot | s_t)$. The higher-level reward r^H typically encodes

progress toward the global objective. Note that due to the dependence of higher-level outcomes on the lower-level policy, the higher-level reward and transitions are non-stationary and *policy-dependent*, i.e., $r_{\pi^L(\cdot|s,g)}^H$.

Lower-Level MDP: Conditioned on the state and subgoal (s_t, g_t) , the lower-level policy $\pi^L : \mathcal{S} \times \mathcal{G} \rightarrow \Delta(\mathcal{A}^L)$ acts in the environment, with an intrinsic reward $r_t^L(s_t, g_t, a_t) = \mathbf{1}_{\{\|s_t - g_t\| < \varepsilon\}}$ for achieving the subgoal, often under a sparse-reward setting. Primitive actions are executed until the subgoal is achieved or an episode horizon reached.

Hierarchical MDP Decomposition. HRL generalizes this structure by introducing policies at multiple levels of abstraction. Following the Feudal architecture [23], we consider a goal-conditioned HRL setup, where the higher-level policy $\pi^H : \mathcal{S} \rightarrow \Delta(\mathcal{G})$ selects a subgoal $g_t \in \mathcal{G}$ every k timesteps, where $\mathcal{G} \subseteq \mathcal{S}$ denotes the set of achievable goals. Specifically, at each decision point t satisfying $t \bmod k = 0$, a new subgoal $g_t \sim \pi^H(\cdot | s_t)$ is sampled and held fixed for the subsequent k steps, i.e., $g_t = g_{t+1} = \dots = g_{t+k-1}$.

During each of these k timesteps, the lower-level policy $\pi^L : \mathcal{S} \times \mathcal{G} \rightarrow \Delta(\mathcal{A})$ receives the current state s_t and the fixed subgoal g_t , and selects a primitive action $a_t \sim \pi^L(\cdot | s_t, g_t)$. The environment then transitions to the next state according to $s_{t+1} \sim p(\cdot | s_t, a_t)$. Higher-level transitions are thus stored as $(s_t, g^*, g_t, r_t^H, s_{t+k})$, and lower-level transitions as $(s_t, g_t, a_t, r_t^L, s_{t+1})$. Through this coordination, the hierarchical agent incrementally works towards achieving the overall task objective.

The higher level Q^H and V^H functions are the expected rewards defined over higher level subgoal prediction trajectories defined in the higher level replay buffer as tuples $(s_t, g^*, g_t, r_t^H, s_{t+k})$ (updated every k timesteps), and the lower level Q^L and V^L functions are the expected rewards defined over lower level primitive action prediction trajectories defined in the lower level replay buffer as tuples $(s_t, g_t, a_t, r_t^L, s_{t+1})$.

HRL: Neural Network Architectures Unless specified otherwise, we use the following neural network architectures in the all subsequent chapters:

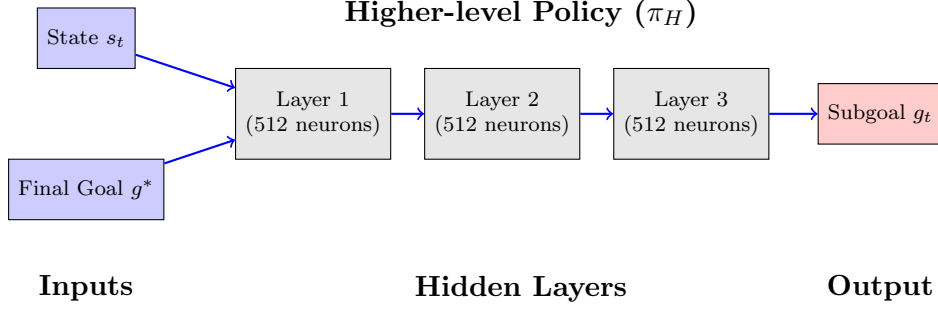


FIGURE 1.1: Higher-level Neural Network Architecture. The network takes state and final goal as inputs and processes them through hidden layers to output subgoals.

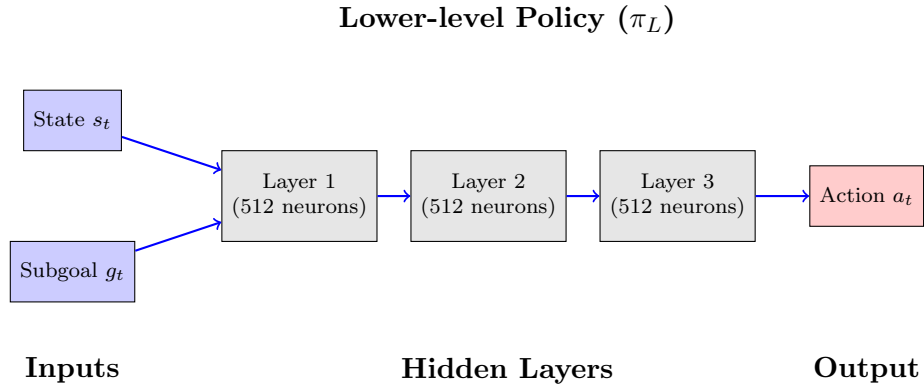


FIGURE 1.2: Lower-level Neural Network Architecture. The network takes state and subgoal as inputs and processes them through hidden layers to output primitive actions.

1.3 Illustrative Example: Robotic Tea-Making.

To concretely illustrate the feudal HRL framework, we consider the robotic tea-making task. This example demonstrates how HRL decomposes complex, long-horizon objectives into coordinated subgoals, each with its own decision-making policy, reward structure, and temporal horizon.

The task *Robotic Tea-Making* can be divided into sub-tasks:

1. Pour water and tea into the kettle.
2. Place the kettle on the gas.
3. Turn on the gas.
4. Place the tea into the cup.

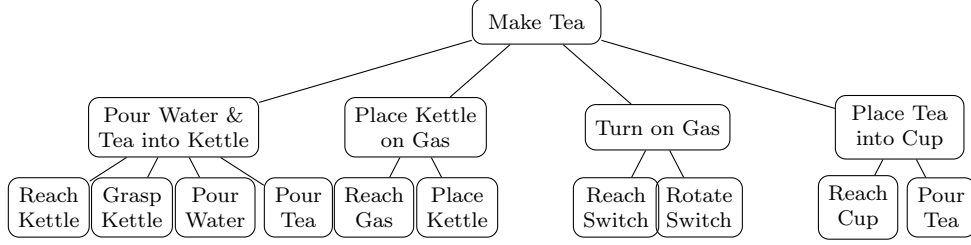


FIGURE 1.3: Hierarchical decomposition of the Robotic Tea-Making task. Each subtask is further divided into manipulation primitives.

We consider a two-level hierarchy, where the higher-level policy (manager) sets subgoals for the lower-level policy (worker) to achieve, each operating at different temporal resolutions. In this setup, the high-level subgoals are systematically decomposed into intermediate objectives, and the lower-level agent executes primitive actions to achieve these subgoals.

Illustrative example: Mathematical Formulation

Illustrative Example: Base MDP Structure. The task is formulated as a Markov Decision Process (MDP) with the tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where the state space $\mathcal{S}$ encompasses robot configuration and environment state. The action space $\mathcal{A}$ includes joint torques and gripper commands, while the transition function p captures the complex dynamics of robot-environment interaction.

Illustrative Example: Higher-Level MDP The higher-level MDP operates over abstract states describing overall task progress. The higher-level policy $\pi^H : \mathcal{S} \rightarrow \Delta(\mathcal{G})$ selects subgoals $g_t \in \mathcal{G}$ every $k = 5$ timesteps, where each subgoal represents a distinct phase of the tea-making process. The higher-level reward structure accumulates external rewards over the temporal window:

$$r_t^H = \sum_{i=kt}^{kt+k-1} r(s_i, a_i)$$

directly reflecting progress toward the ultimate goal of making tea. This reward structure incentivises the higher-level policy to select subgoals that advance the overall task while maintaining efficiency and safety constraints.

Illustrative Example: Lower-Level MDP The lower-level policy $\pi^L : \mathcal{S} \times \mathcal{G} \rightarrow \Delta(\mathcal{A})$ operates on the full robot state space, including joint angles, end-effector positions and torques. This policy receives subgoals from the higher level and executes primitive actions to achieve them, with intrinsic rewards based on subgoal proximity:

$$r_t^L = \begin{cases} 1 & \text{if } \|g_t - s_{t+1}\|_2 < \varepsilon \\ 0 & \text{otherwise} \end{cases}$$

Illustrative Example: Task Learning The learning process in this hierarchical framework may leverage reinforcement learning (RL) at multiple levels of abstraction to efficiently solve the overall task. In the tea-making example, the higher-level policy (manager) is trained with RL to select effective sequences of subgoals, using sparse rewards that reflect successful completion of major task milestones (such as boiling water or pouring tea). The lower-level policy (worker) learns to achieve each subgoal by directly interacting with the environment and receiving intrinsic, sparse rewards for achieving the subgoals.

During training, the manager proposes a subgoal, and the worker attempts to achieve it by executing primitive actions. If the worker reaches the designated subgoal, they receive a positive intrinsic reward, reinforcing desired behaviour. Simultaneously, the manager observes accumulated extrinsic rewards and refines its high-level planning. By separating learning into distinct yet interdependent layers, the hierarchical framework drastically improves exploration, credit assignment, and skill reuse [6], enabling the agent to efficiently solve complex, long-horizon tasks.

Illustrative Example: Summary This robotic tea-making example concretely demonstrates the hierarchical framework application. By explicitly decomposing a long-horizon, complex task into sequential subgoals, the hierarchical approach enables each level of the agent to operate at an appropriate temporal and semantic scale. The higher-level manager focuses on abstract, goal-oriented planning, while the lower-level worker specialises in executing detailed, short-horizon actions to realise specific subgoals. This clear abstraction not only simplifies policy learning and credit assignment but also reflects how humans solve

intricate problems by dividing them into manageable subtasks. Ultimately, this structure makes learning more sample-efficient, robust, and scalable for real-world tasks that are otherwise intractable for flat, non-hierarchical agents.

1.4 HRL: Design Challenges

Hierarchical Reinforcement Learning (HRL) provides powerful mechanisms for solving complex, temporally extended tasks by decomposing them into subproblems. However, the introduction of hierarchy also brings a distinct set of design and practical challenges that can limit its effectiveness. Below, we outline the major design challenges that arise when building and applying HRL systems.

1. Non-Stationarity As discussed in [26] and [27], in hierarchical reinforcement learning (HRL), updates to the lower-level policy π^L inevitably alter its behaviour over time. Because the higher-level policy relies on the lower level policy π^L , any change to the lower-level policy effectively modifies the transition dynamics and reward function experienced by the higher-level policy. As a result, the higher-level decision process becomes *non-stationary*, undermining the stability and reliability of hierarchical learning and potentially leading to unstable or suboptimal learning outcomes. Notably, this occurs when the lower-level policy is sub-optimal. Since, intuitively, in such a case, even an optimal higher-level subgoal may lead to poor rewards.

Transition Function Non-stationarity: Mathematical illustration Let the high-level transition distribution be

$$p^H(s_{t+k} \mid s_t, g_t) = \prod_{i=0}^{k-1} p(s_{t+i+1} \mid s_{t+i}, a_{t+i}), \text{ where,} \quad (1.2)$$

$$a_{t+i} \sim \pi^L(\cdot \mid s_{t+i}, g_t) \text{ for } i = 0, \dots, k-1. \quad (1.3)$$

If $\pi^L \rightarrow \pi^{L'}$, then,

$$p^H(s_{t+k} \mid s_t, g_t; \pi^L) \neq p^H(s_{t+k} \mid s_t, g_t; \pi^{L'}). \quad (1.4)$$

This implies that with changing lower-level policy, the higher-level transition function becomes non-stationary.

2. Infeasible Subgoals Generation A central promise of HRL is that the higher-level policy will propose meaningful, achievable subgoals for the lower-level policy. As observed in [28], however, the manager often generates subgoals that are infeasible for the lower-level policy to achieve. This leads to unstable learning, since even the optimal lower-level policy is unable to achieve the predicted subgoals.

3. Subgoal and Skill Discovery: Discovering effective subgoals and reusable skills is a central, yet unresolved, issue in hierarchical RL. Fixed subgoal representations may not be suitable or generalizable for many scenarios. Several algorithms targeted bottleneck states, subgoal discovery [24, 29, 30], or maximization of coverage/diversity in behaviors [31, 32]. However, such subgoals may be sub-optimal, especially in sparse long horizon settings that require extensive exploration. Moreover, autonomously learning appropriate subgoal representations is a difficult open research problem, often requiring meta-learning or additional heuristics. The optimal subgoal structure is typically domain-dependent, making transfer across environments challenging. In this thesis, we propose several methods for efficient and autonomous skill or subgoal discovery, which does not require a domain expert.

4. Coordination and Cooperation Between Hierarchical Levels: For HRL to work effectively, the policies at different levels must cooperate effectively with each other in order to collectively solve the task. Otherwise, it may lead to degenerate solutions, thus undermining the advantages of the hierarchical framework. Ideally, the task split should be balanced between different hierarchical levels. Notably, this is a very important challenge with HRL, especially when the hierarchical levels are learned in parallel.

1.5 Proposed Solution

To address the core challenges inherent in hierarchical reinforcement learning (HRL), this thesis proposes a comprehensive suite of algorithmic innovations and frameworks, each designed to target specific obstacles identified in the field. Building on the observation that hierarchical abstraction is fundamental to human intelligence, the solutions presented aim to empower robotic agents to tackle complex, long-horizon tasks—particularly those plagued by sparse rewards and difficult credit assignment—by leveraging the strengths of hierarchical decomposition and informed supervision.

1.5.1 Developing evaluation benchmarks

In order to develop efficient and robust algorithms for solving complex, long-horizon tasks, it is crucial to establish effective evaluation benchmarks that support both quantitative and qualitative analyses of the proposed approaches. These benchmarks should allow for rigorous testing and comparison against state-of-the-art baselines in real world and challenging long horizon environments. Hence, in Part I Chapter 4, we first design and develop robust robotic environments in both simulation and the real world, focusing on complex tasks pertaining to robotic navigation and manipulation. These task environments provide a platform for evaluating the performance and generalization of hierarchical reinforcement learning (HRL) algorithms under diverse scenarios.

1.5.2 Leveraging expert demonstrations

In Part II, we train efficient hierarchical policies by leveraging a few expert demonstrations to generate efficient subgoal supervision for the higher-level policy, and thus propose stable HRL algorithms which mitigate non-stationarity in HRL. In Chapter 5, we propose CRISP, a novel HRL algorithm that effectively generates a curriculum of achievable subgoals for evolving lower-level primitives using reinforcement learning and imitation learning. CRISP uses the lower level primitive to periodically perform data relabeling on a handful of

expert demonstrations, using a novel primitive informed parsing (PIP) approach, thereby mitigating non-stationarity. Further, in Chapter 6, we propose primitive enabled adaptive relabeling (PEAR), a two-phase approach which first performs adaptive relabeling on a few expert demonstrations to generate efficient subgoal supervision, and then jointly optimizes HRL agents by employing reinforcement learning (RL) and imitation learning (IL).

1.5.3 Leveraging preference-based learning

We then tackle scenarios where acquiring expert demonstrations is non-trivial. To this end, in Part III, we tackle inherent limitations of HRL by leveraging advancements in preference-based learning [33, 34], thereby enabling the higher-level policy to refine its decision making ability through human-like preferences. In Chapter 7, we introduce PIPER, primitive-informed preference-based hierarchical reinforcement learning via hindsight relabeling, a novel approach that leverages preference-based learning to learn a reward model, and subsequently uses this reward model to relabel higher-level replay buffers. Further, in Chapter 8, we introduce hierarchical preference optimization (HPO), a novel approach to hierarchical reinforcement learning (HRL) that leverages maximum entropy reinforcement learning combined with token-level direct preference optimization (DPO) [34], for learning efficient control policies to perform complex robotic tasks, especially in sparse reward settings.

1.5.4 Leveraging natural language instructions

Using language instructions to directly guide and control robots has long been a cherished goal in the field of robotics. What if we could instruct robots by describing the target task in natural language, allowing the robot to handle the low-level actions required to accomplish the task? To this end, in Part IV Chapter 9, we develop an efficient HRL framework that translates natural language instructions directly into lower level robotic control actions. This approach leverages the power of language as a guiding signal, enabling robots to interpret and execute complex tasks described in natural language.

Finally, Chapter 10 concludes by summarizing the proposed concepts, algorithms, and results, while also outlining potential future directions that stem from this work. Collectively, these solutions form a unified and versatile toolkit tailored to overcome the most pressing challenges in HRL, namely, non-stationarity, subgoal feasibility, skill discovery, and inter-level coordination. Throughout the thesis, each proposed strategy is thoroughly evaluated and contextualized, highlighting their roles in advancing both the theoretical robustness and practical deployment of hierarchical reinforcement learning systems.

Chapter 2

Related Work

2.1 Advancements in Dynamic Programming, Artificial Intelligence, and Machine Learning

The foundations of dynamic programming, artificial intelligence, and machine learning have undergone remarkable development over the past century, laying the groundwork for contemporary hierarchical learning frameworks.

2.1.1 Dynamic Programming.

The field of dynamic programming originated with the pioneering contributions of Richard Bellman in the 1950s [35, 36]. Dynamic programming (DP) provides a rigorous mathematical framework for solving sequential decision-making problems under uncertainty, allowing for the computation of optimal policies by recursively breaking down problems into smaller subproblems. This principle has profoundly influenced stochastic control, operations research, and optimal planning. Despite its theoretical power, classical DP methods suffer from the “curse of dimensionality,” as computational complexity grows exponentially with the size of the state and action spaces, making them impractical for many real-world applications.

2.1.2 Artificial Intelligence.

During the second half of the 20th century, artificial intelligence (AI) emerged as a major scientific discipline with the vision of creating machines capable of general problem-solving and reasoning [37, 38]. Early AI research focused on symbolic reasoning, search algorithms, and the development of expert systems. However, scaling such approaches to complex, uncertain environments proved to be a significant challenge, spurring the exploration of more adaptive, data-driven methods.

2.1.3 Machine Learning and Reinforcement Learning.

The field of machine learning built on probabilistic modeling, statistical pattern recognition, and computational learning theory to enable agents to acquire knowledge from data [39, 40]. Over the past few years, machine learning has led to transformative advancements that have been able to reshape numerous fields. In early 2010s, deep learning based approaches became popular, led by advancements in convolutional neural networks (CNNs) for image recognition and computer vision tasks (with the success of AlexNet in the ImageNet competition [41]). This era also saw the development of recurrent neural networks (RNNs) and their enhancements like LSTMs and GRUs, which improved sequential data processing in areas such as natural language processing (NLP) and time-series analysis. This was followed by the advancements in generative modeling, such as GANs (Generative Adversarial Networks) and VAEs (Variational Autoencoders) which revolutionized generative AI, enabling applications like Stable Diffusion [42]. However, the introduction of the transformer architecture [43] marked a paradigm shift, leading to state-of-the-art models like BERT and GPT [44], which excel in efficient natural language generative models or foundation models.

In particular, reinforcement learning (RL) formalized the problem of learning to act via reward-driven interaction with the environment [11, 45]. Breakthroughs in RL, such as value iteration [35], temporal difference learning [46], and policy gradient methods [47], bridged control theory and AI, while deep learning architectures enabled RL agents to

learn effective representations directly from raw sensory input [48, 49]. Landmark achievements—such as programs mastering board games [1], video games [49], and robotic control [4] demonstrate the capacity of modern RL to address increasingly complex tasks.

Hierarchical reinforcement learning (HRL) builds upon the advancements in artificial intelligence, operations research, robotics, and cognitive science. Its history extends back several decades to early conceptualizations of abstraction and hierarchy for effective decision making, planning, and control. We now provide a comprehensive review, spanning from foundational works to the most recent advances in deep hierarchical reinforcement learning and related cross-disciplinary areas.

2.2 Foundations of Hierarchical Control and Temporal Abstraction

The motivation for HRL is rooted in early observations from cognitive psychology and neuroscience, where humans and animals naturally decompose long-horizon tasks into reusable subtasks or skills [14, 15]. In artificial intelligence, planning with abstraction and sub-goals has been extensively studied, with classical approaches in hierarchical planning and task decomposition [50–53].

Modern HRL formally began with frameworks designed to introduce temporal abstractions into reinforcement learning agents. The *Options* framework [12] allows agents to learn and utilize temporally extended actions, or “options”, defined by initiation and termination criteria, greatly enabling multi-time-scale reasoning. The *Hierarchy of Abstract Machines* (HAMs) [54] and *feudal reinforcement learning* [23] both introduced architectures in which high-level controllers delegate responsibility to lower-level sub-managers, organizing policies as multilevel hierarchies. A foundational advance came with Dietterich’s MAXQ value function decomposition [55], which demonstrated how a complex RL problem can be hierarchically decomposed into a set of subproblems, each equipped with its own value function

and local policies. MAXQ provides both a declarative semantics (a recursive value function decomposition) and procedural semantics (a subroutine hierarchy).

Importantly, these early HRL approaches were deeply influenced by advances in fields such as cognitive science and computational neuroscience, which highlighted hierarchical processing and control in natural agents [14, 15].

2.2.1 Early Hierarchical RL in Practice

The 1990s and early 2000s witnessed pioneering practical HRL implementations, such as learning macro-actions in gridworlds [29, 56] and robot navigation and manipulation. These works demonstrated how temporal abstraction and skill reuse could improve exploration, learning efficiency, and credit assignment in environments with sparse, long-term rewards. Surveys such as [57] and [58] provide an overview of these formative years of research.

2.2.2 Hierarchical RL and Planning in Decision Making and Robotics

Much research has focused on leveraging hierarchical planning for decision making and autonomous systems. Hierarchical planners such as STRIPS variants [50], HTN planning, and multi-level control schemes [52] influenced subsequent robot architectures. In reinforcement learning, researchers extended options and HAMs to real-world control [59], hierarchical actor-critic designs [24, 60], and skill discovery algorithms [29, 61].

Notably, the challenge of designing or manually specifying the decomposition of tasks and appropriate subtasks has been a recurring theme, motivating subsequent work on automatic skill discovery [29, 62] and abstraction from data.

2.2.3 Hierarchical RL in Cognitive Science and Neuroscience

Beyond engineering, HRL models were inspired from cognitive neuroscience, offering computational explanations for how the brain organizes complex behavior via cascading control and motivation at different levels of abstraction [15, 63]. These insights have informed architectures in artificial agents and inspired behavioral priors for learning.

2.2.4 Automatic Discovery of Temporal Abstractions and Options

A notable line of work investigated how agents could autonomously discover useful temporally extended actions or options. Several algorithms targeted bottleneck states, subgoal discovery [24, 29, 30], or maximization of coverage/diversity in behaviors [31, 32]. The principle is to facilitate exploration and learning by enabling agents to identify reusable skills that correspond to important events or transitions in the environment.

2.3 Recent Advances: Deep HRL and Large-Scale Autonomous Agents

The advent of deep learning reinvigorated HRL research, enabling application in high-dimensional visual and robotic domains. Modern breakthrough architectures include Option-Critic [64], Feudal Networks [65], HIRO [66], and HAC [67]. These algorithms explicitly address challenges such as non-stationarity and credit assignment in concurrently trained hierarchies, and many employ goal-conditioned value functions.

In *goal-conditioned* hierarchical framework, the high-level policy predicts subgoals for lower primitive, which in turn performs primitive actions directly on the environment [65–67]. This hierarchical framework allows agents to tackle long-horizon tasks by breaking them into smaller, manageable subtasks, enabling more effective exploration and better credit assignment across extended time scales.

Concurrently, hierarchical approaches have proven essential in large-scale robotic manipulation, navigation, and language-conditioned control, facilitating the composition of complex tasks from learned skills, and integrating learning from demonstration, imitation, and curriculum learning [68–71]. Recent advances also investigate language-to-action and code via large language models (LLMs) and corresponding hierarchical architectures for autonomous agents.

2.4 Summary

The field of hierarchical reinforcement learning has evolved substantially, building on decades of progress across artificial intelligence, robotics, and cognitive science. Table 2.1 provides an overarching view of key milestones that have shaped the trajectory of HRL research over the past several decades.

TABLE 2.1: Key milestones in Hierarchical Reinforcement Learning

Year	Contribution
1970s-80s	Hierarchical planning, abstraction in AI and robotics
1990s	Options, HAMs, Feudal RL, MAXQ, practical HRL applications
2000s	Skill discovery, abstraction learning, cognitive science models
2010s	Deep HRL, goal-conditioned RL
2020s	Language-conditioned HRL, LLMs, large-scale robotics

2.5 Open Problems and Future Directions

Despite steady progress, open challenges in HRL still remain:

1. automatic skill discovery and autonomous subgoal generation.
2. efficiently breaking down tasks while avoiding degenerate solutions.
3. improving generalization in HRL via skill-reuse.

Further advances in these directions are required for building scalable and general-purpose artificial agents.

Chapter 3

Problem Definitions

This chapter formalizes the main mathematical structures and challenges underlying the methods and settings addressed in this thesis. We focus on the formal definitions of Markov Decision Processes, goal-conditioned RL, the hierarchical RL (HRL) framework, preference-based learning, and language to rewards translation. These problem definitions establish the core algorithmic and theoretical landscape for the solutions and analyses that follow in subsequent chapters.

3.1 Markov Decision Processes (MDP)

We begin with the formalization of a Markov Decision Process (MDP), defined as the tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where:

- $\mathcal{S}$: state space
- $\mathcal{A}$: action space
- $p : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$: transition probability function
- $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$: reward function
- $\gamma \in (0, 1)$: discount factor

A policy $\pi : \mathcal{S} \rightarrow \Delta(\mathcal{A})$ models action selection probabilities in each state. At each timestep t , the agent observes s_t , samples $a_t \sim \pi(\cdot|s_t)$, receives reward $r_t = r(s_t, a_t)$, and transitions to $s_{t+1} \sim p(\cdot|s_t, a_t)$.

The classical RL objective is to find an optimal policy maximizing the expected discounted return:

$$\pi^* = \arg \max_{\pi} J(\pi), \quad \text{where} \quad J(\pi) = \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right] \quad (3.1)$$

The state value and state-action value functions are defined as:

$$V_{\pi}(s) = \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s \right] \quad (3.2)$$

$$Q_{\pi}(s, a) = \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a \right] \quad (3.3)$$

3.2 Goal-Conditioned Reinforcement Learning

For tasks with explicit goal specifications, we adopt the Universal MDP (UMDP) [72] formalism, given by the tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma, \mathcal{G})$, where $\mathcal{G}$ is the goal space. Each episode samples $g \sim \mathcal{G}$; the agent follows a goal-conditioned policy $\pi(a|s, g)$. The objective is to maximize:

$$J(\pi) = (1 - \gamma)^{-1} \mathbb{E}_{s \sim d^{\pi}, a \sim \pi(a|s, g), g \sim \mathcal{G}} [r(s, a, g)] \quad (3.4)$$

where $d^{\pi}(s)$ is the discounted stationary state distribution under π . In sparse reward settings, agents are frequently rewarded by proximity to the goal, e.g.,

$$r(s, a, g) = \mathbf{1}_{\{\|s' - g\|_2 \leq \epsilon\}} \quad (3.5)$$

where s' is the next state and ϵ is a proximity threshold.

3.3 Hierarchical Reinforcement Learning (HRL)

To address long-horizon and compositional tasks, we formalize a goal-conditioned, bi-level hierarchical RL framework.

3.3.1 Hierarchical Policy Decomposition

- **Higher-Level Policy:** $\pi^H : \mathcal{S} \times \mathcal{G} \rightarrow \Delta(\mathcal{G})$ proposes subgoals $g_t \in \mathcal{G} \subseteq \mathcal{S}$ every k timesteps based on state s_t and final goal g^* .
- **Lower-Level Policy:** $\pi^L : \mathcal{S} \times \mathcal{G} \rightarrow \Delta(\mathcal{A})$ generates primitive actions to achieve the subgoal g_t within k timesteps.

3.3.2 Execution Protocol

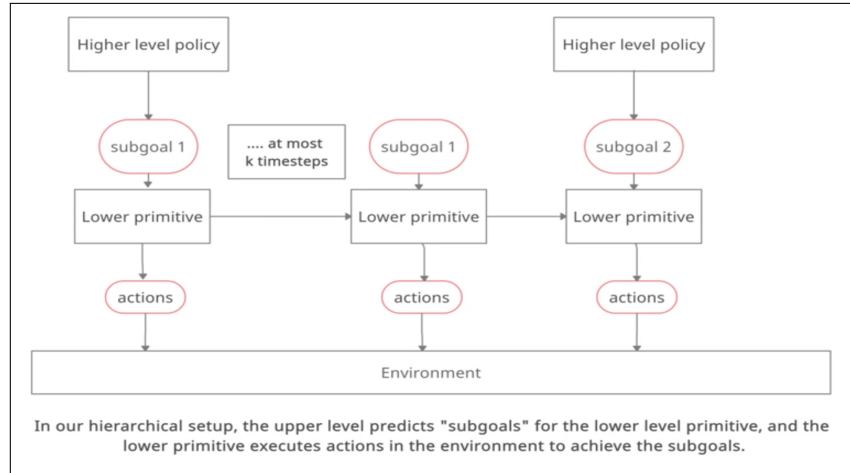


FIGURE 3.1: Execution Protocol.

At each timestep t :

- If $t \bmod k = 0$, the higher-level policy selects $g_t \sim \pi^H(\cdot | s_t, g^*)$
- Otherwise, subgoal remains fixed: $g_t = g_{k \cdot \lfloor t/k \rfloor}$
- The lower-level policy selects $a_t \sim \pi^L(\cdot | s_t, g_t)$
- System transitions: $s_{t+1} \sim p(\cdot | s_t, a_t)$

3.3.3 Reward Structure and Buffers

- **Lower-Level Reward:**

$$r_t^L = \mathbf{1}_{\{\|s_{t+1}-g_t\|_2 \leq \epsilon^L\}} - 1$$

- **Higher-Level Reward:**

$$r_t^H = \mathbf{1}_{\{\|s_{t+1}-g^*\|_2 \leq \epsilon^H\}} - 1$$

- **Experience Storage:**

- Lower level: $(s_t, g_t, a_t, r_t^L, s_{t+1})$
- Higher level: $(s_t, g^*, g_t, \sum_{i=t}^{t+k-1} r_i^H, s_{t+k})$ (updated every k steps)

3.3.4 Key Challenges in HRL

1. Non-Stationarity: As the lower-level policy $\pi_L^{(m)}$ evolves over the course of training (with m indexing training iteration), the conditional distribution over next states s_{t+k} after executing a fixed high-level subgoal g_t changes:

$$p_{g_t}^{(m)}(s_{t+k} \mid s_t) = \Pr(s_{t+k} \mid s_t, g_t, \pi_L^{(m)}).$$

Accordingly, the cumulative high-level reward over the k -step interval,

$$r_t^{H(m)} = r^H(s_t, g^*, g_t; \pi_L^{(m)}),$$

becomes non-stationary, as it depends on the continually adapting lower-level behavior. This drift invalidates older transitions stored in the higher-level replay buffer, undermining the stability and effectiveness of off-policy learning for the higher-level policy π_H and substantially impeding convergence.

$$p^H(s_{t+k}|s_t, g_t; \pi^L) \neq p^H(s_{t+k}|s_t, g_t; \pi^{L'}) \quad (3.6)$$

Since higher-level transitions depend on a sequence of lower-level actions, stability is difficult when π^L is not stationary.

2. Infeasible Subgoal Generation. The higher-level policy may propose subgoals that the current lower-level policy cannot achieve within k timesteps, causing poor higher-level learning and insufficient exploration.

3. Coordination. Effective HRL hinges on balanced decomposition between levels to avoid degenerate solutions (e.g., trivial or irrelevant subgoals).

3.4 Preference-Based Reinforcement Learning

When explicit reward engineering is costly or ill-defined, preference-based RL (PbRL) uses pairwise or groupwise trajectory comparisons, typically provided by humans, as the learning signal.

3.4.1 Bradley-Terry Model

Given two trajectories τ^1, τ^2 , let $\hat{r}_\phi$ denote a learned reward model. The probability that τ^1 is preferred over τ^2 is modeled as:

$$P_\phi(\tau^1 \succ \tau^2) = \frac{\exp(\sum_t \hat{r}_\phi(s_t^1, a_t^1))}{\exp(\sum_t \hat{r}_\phi(s_t^1, a_t^1)) + \exp(\sum_t \hat{r}_\phi(s_t^2, a_t^2))} \quad (3.7)$$

3.4.2 RLHF: Reinforcement Learning from Human Feedback

Three main steps:

1. **Preference Collection:** Gather $\mathcal{D} = \{(\tau^1, \tau^2, y)\}$

2. **Reward Modeling:** Train $\hat{r}_\phi$ by maximum likelihood:

$$\mathcal{L}(\phi) = -\mathbb{E}_{(\tau^1, \tau^2, y) \sim \mathcal{D}} \left[y \log \sigma \left(\sum_t \hat{r}_\phi(s_t^1, a_t^1) - \sum_t \hat{r}_\phi(s_t^2, a_t^2) \right) \right] \quad (3.8)$$

3. **Policy Optimization:** Optimize the RL policy w.r.t. the learned reward.

3.4.3 Direct Preference Optimization (DPO)

Instead of learning a reward model, DPO [34] directly optimizes the policy based on preference data:

$$\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}_{(\tau^1, \tau^2, y) \sim \mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(\tau^1)}{\pi_{ref}(\tau^1)} - \beta \log \frac{\pi_\theta(\tau^2)}{\pi_{ref}(\tau^2)} \right) \right] \quad (3.9)$$

where β is a temperature, π_θ is the target policy, and π_{ref} is a reference policy.

3.4.3.1 Limitations of directly applying PbL to HRL

Directly using RLHF: Prior approaches may lead to degenerate solutions by generating infeasible subgoals for the lower-level primitive. Additionally, such approaches require RL as an intermediate step, which might cause training instability [34].

Directly using DPO: In temporally extended task environments like robotics, directly extending DPO to the HRL framework is also non-trivial due to three reasons: (i) such scenarios deal with multi-step trajectories involving stochastic transitions models, (ii) efficient pre-trained reference policies are typically unavailable in robotics, (iii) similar to RLHF, such approaches may produce degenerate solutions when the higher level policy subgoal predictions are infeasible.

3.5 Language to Rewards (L2R)

Recent research has established methods for translating natural language instructions into reward signals by leveraging domain-specific reward models [73–75]. Of particular relevance is the L2R framework [76], which utilizes language instructions to parameterize rewards that serve as the interface between high-level linguistic goals and low-level robotic actions. In L2R, a specialized reward translator module converts natural language descriptions into reward parameters, which are then optimized—originally using a MuJoCo model predictive controller (MPC) [77]—in order to accomplish diverse robotic tasks.

In this thesis, we move beyond MPC-based optimization in L2R and employ reinforcement learning (RL) to jointly optimize both higher- and lower-level policies using the generated reward parameters.

3.5.1 Limitations of L2R framework

Despite its promise, the original L2R framework has two important limitations:

- L1:** For complex, long-horizon tasks, relying solely on a single-level policy makes learning intractable or ineffective.
- L2:** The reward functions generated can be excessively sparse, impeding progress and learning of lower-level primitives.

3.6 Problem Statement

The overarching goal of this thesis is to develop robust hierarchical reinforcement learning methods that:

- **Mitigate Non-Stationarity:** Address instability caused by evolving lower-level policies in off-policy HRL

- **Ensure Feasible Subgoal Generation:** Prevent higher-level subgoal proposals that cannot be achieved by the lower-level policy
- **Enable Effective Coordination:** Balance the cooperation between hierarchical levels to avoid degenerate behaviors
- **Leverage Supervision:** Utilize scarce expert demonstrations, preference feedback or language-guided learning to guide learning
- **Scale to Complex Domains:** Solve long-horizon, sparse-reward robotic or sequential tasks in challenging domains

Subsequent chapters address these challenges using demonstration and preference-based guidance, language-driven reward specification, and improved algorithms for coordination and stability in HRL.

List of submissions related to thesis

- **Utsav Singh**, Vinay P. Namboodiri “PEAR: Primitive enabled Adaptive Relabeling for boosting Hierarchical Reinforcement Learning.” Accepted at ICLR 2025 [78].
- **Utsav Singh**, Wesley A. Suttle, Brian M. Sadler, Vinay P. Namboodiri, Amrit Singh Bedi “PIPER: Primitive-Informed Preference-based Hierarchical Reinforcement Learning via Hindsight Relabeling.” Accepted at ICML 2024 [79].
- **Utsav Singh**, Souradip Chakraborty, Wesley A. Suttle, Brian M. Sadler, Anit Kumar Sahu, Mubarak Shah, Vinay P. Namboodiri, Amrit Singh Bedi “Hierarchical Preference Optimization: Learning to achieve goals via feasible subgoals prediction.” Under review at NeurIPS 2025 [80].
- **Utsav Singh**, Prमित Bhattacharyya, Vinay P. Namboodiri “LGR2: Language Guided Reward Relabeling for Accelerating Hierarchical Reinforcement Learning.” Under review at AAAI 2026 [81].
- **Utsav Singh**, Vinay P. Namboodiri “CRISP: Curriculum Inducing Primitive Informed Subgoal Prediction for Hierarchical Reinforcement Learning.” Under review at AAAI 2026 [82].

Part I

Designing Evaluation Frameworks for Complex Control Tasks

Chapter 4

Robust Evaluation Frameworks for Complex Control Tasks

You cannot improve what you do not measure.

– Peter Drucker

To develop efficient and robust algorithms for solving complex, long-horizon tasks, it is essential to have effective evaluation benchmarks that enable both quantitative and qualitative analysis of the proposed approaches and facilitate comparisons with state-of-the-art baselines. Notably, short-horizon environment tasks require minimal interactions with the environment and can often be addressed using flat (single-level) approaches. However, as demonstrated later in this thesis, complex tasks that necessitate long-term interactions with the environment introduce significant challenges, such as credit assignment and exploration, thereby rendering single-level approaches ineffective. In contrast, hierarchical approaches leverage temporal abstraction and improved exploration [6], and are thus better suited to solving such complex, long-horizon tasks. Also, in this thesis, we focus on sparse reward environments, where the agents are rewarded only when achieving the final objective. Such sparse environments require complex long term planning, and are therefore ideal test-beds for evaluating long horizon strategies.

Although prior works propose a number of evaluation environments in robotics scenarios [83–87], most such environments are effectively short horizon environments. Hence, effective evaluation of hierarchical strategies necessitates the development of complex long horizon environments. These environments should possess the following key properties:

1. The environments should have long horizons, requiring agents to engage extensively with the environment to successfully solve the task.
2. The environment tasks should be divisible into shorter subtasks that can be combined to solve the complete task.
3. The environments should be sparsely rewarded environments.
4. The environments should adhere to the laws of physics and maintain a high degree of accuracy.
5. The environments should be computationally efficient.

To this end, we first develop six such robotics environments in simulation using Mujoco [88] simulator, and then develop the corresponding real world robotics environments for effective simulation to real translation of learnt policies:

1. Randomized Maze Navigation Environment.
2. Pick and Place Environment.
3. Push Environment.
4. Bin Environment.
5. Hollow Environment.
6. Rope Manipulation Environment.

Additionally, we also utilize franka kitchen [89] environment for our evaluations. In prior work [89], this environment is densely rewarded, and therefore we modify it to handle

sparsely rewarded tasks. In the next section, we define the tasks and environments in detail, and provide their relevant implementation details.

4.1 Environments and their implementation details

In this section, we provide the environment and implementation details for all the tasks.

4.1.1 Randomized maze navigation task

In this environment, a 7-DOF robotic arm gripper navigates across random four room mazes to reach the goal position. In this task, the gripper (whose height is kept fixed at table height) has to navigate across randomly generated four room mazes (the wall and gate positions are randomly generated) to achieve final goal. In the pick and place task, the gripper has to pick a randomly placed square block and bring it to a randomly generated goal position. The gripper arm is kept closed and fixed at table height, and the positions of walls and gates are randomly generated. The table is discretized into a rectangular $W * H$ grid, and the vertical and horizontal wall positions W_P and H_P are randomly picked from $(1, W - 2)$ and $(1, H - 2)$ respectively. In the four room environment thus constructed, the four gate positions are randomly picked from $(1, W_P - 1)$, $(W_P + 1, W - 2)$, $(1, H_P - 1)$ and $(H_P + 1, H - 2)$.

In the maze environment, the state is represented as the vector $[dx, M]$, where dx is current gripper position and M is the sparse maze array. The higher level policy input is thus a concatenated vector $[dx, M, g]$, where g is the target goal position, whereas the lower level policy input is concatenated vector $[dx, M, s_g]$, where s_g is the sub-goal provided by the higher level policy. M is a discrete $2D$ one-hot vector array, where 1 represents presence of a wall block, and otherwise. The lower primitive action a is a 4 dimensional vector with every dimension $a_i \in [0, 1]$. The first 3 dimensions provide offsets to be scaled and added to gripper position for moving it to the intended position. The last dimension provides

gripper control(0 implies a closed gripper and 1 implies an open gripper). We select 100 randomly generated mazes each for training, testing and validation.

Here is a qualitative visualization of a successful attempt at performing the task:

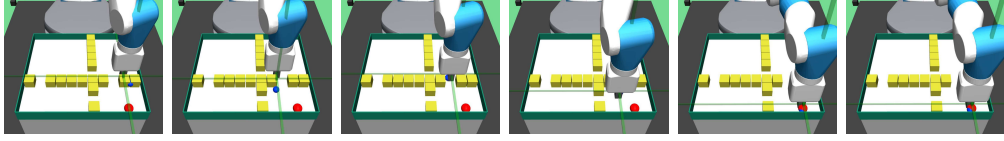


FIGURE 4.1: **Maze navigation task visualization:** The visualization is a successful attempt at performing maze navigation task.

4.1.2 Pick and place, push, bin and hollow environments

In this section, we explain the environment details for the pick and place, push, bin and hollow tasks. The robotic pick and place, bin, and hollow environments employ a 7-DOF robotic arm gripper. In the pick and place task, the gripper has to pick a randomly placed square block and bring it to a randomly generated goal position. In the push environment, the gripper must push a square block towards the goal position on the table. In bin environment, the gripper has to pick up the block and place it in a specific bin placed on the table. In the hollow task, the gripper has to pick a square hollow block and place it across a fixed vertical pole on the table such that the block goes through the pole.

The state is represented as the vector $[dx, o, q, e]$, where dx is the current gripper position, o is the position of the block object placed on the table, q is the relative position of the block with respect to the gripper, and e consists of linear and angular velocities of the gripper and the block object. The higher level policy input is thus a concatenated vector $[dx, o, q, e, g]$, where g is the target goal position. The lower level policy input is concatenated vector $[dx, o, q, e, s_g]$, where s_g is the sub-goal provided by the higher level policy. In our experiments, we keep the sizes of dx , o , q , e to be 3, 3, 3 and 11 respectively. The lower primitive action a is a 4 dimensional vector with every dimension $a_i \in [0, 1]$. The first 3 dimensions provide gripper position offsets, and the last dimension provides gripper control. While training, the position of block object and goal are randomly generated

(block is always initialized on the table, and goal is always above the table at a fixed height). We select 100 random each for training, testing and validation.

Here is a qualitative visualization of a successful attempt at performing the pick and place task:

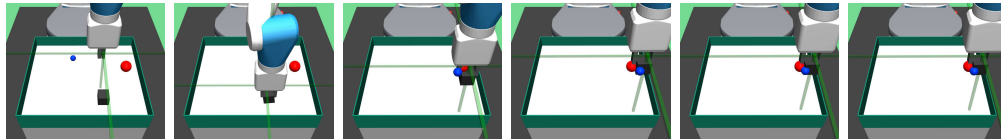


FIGURE 4.2: **Pick and place task visualization:** The visualization is a successful attempt at performing pick and place task.

Here is a qualitative visualization of a successful attempt at performing the push task:

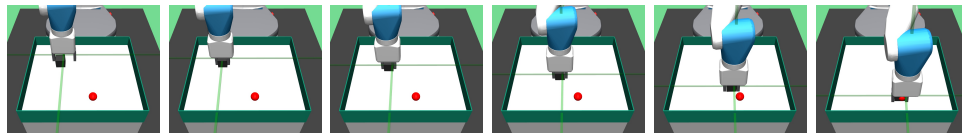


FIGURE 4.3: **Push task visualization:** The visualization is a successful attempt at performing push task.

Here is a qualitative visualization of a successful attempt at performing the bin task:

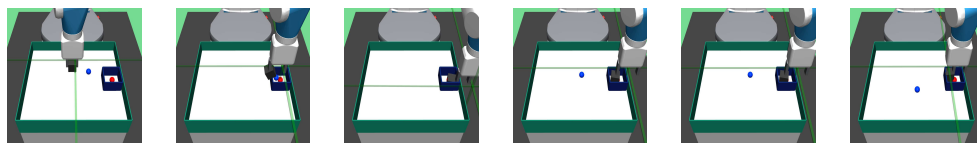


FIGURE 4.4: **Bin task visualization:** The visualization is a successful attempt at performing bin task.

Here is a qualitative visualization of a successful attempt at performing the hollow task:

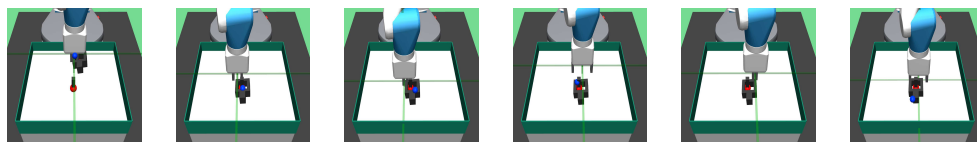


FIGURE 4.5: **Hollow task visualization:** The visualization is a successful attempt at performing hollow task.

4.1.3 Rope Manipulation Environment

The robotic rope environment employ a 7-DOF robotic arm gripper. In rope manipulation task, a deformable soft rope is kept on the table and the gripper performs pokes on the rope to nudge it towards the goal rope configuration. The rope manipulation task involves learning challenging dynamics and goes beyond prior work on navigation-like tasks where the goal space is limited. The deformable rope is formed from 15 constituent cylinders joined together. The state space for the rope manipulation environment is a vector formed by concatenation of the intermediate joint positions. The upper level predicts subgoal s_g for the lower primitive. The action space of the poke is (dx, dy, η) , where (x, y) is the initial position of the poke, and η is the angle describing the direction of the poke. We fix the poke length to be 0.08. We select 100 randomly generated initial and final rope configurations each for training, testing and validation.

Here is a qualitative visualization of a successful attempt at performing the task:

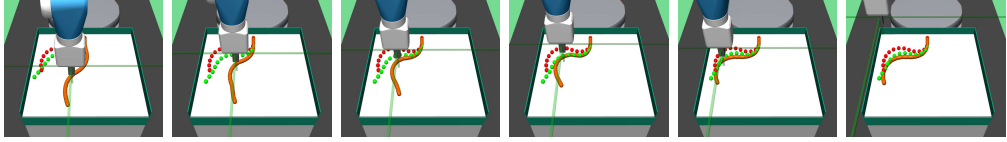


FIGURE 4.6: **Rope manipulation task visualization:** The visualization is a successful attempt at performing rope manipulation task.

4.1.4 Franka kitchen Environment

The franka kitchen environment employs a 9-DoF franka robot gripper. We leverage the D4RL benchmark[90] for this environment, and modify the environment as a sparsely rewarded environment for multiple tasks. As an example, the franka robot has to perform a complex multi-stage task in order to achieve the final goal: the gripper has to first open the microwave door, and then switch on the specific gas knob where the kettle is placed.

Here is a qualitative visualization of a successful attempt at performing the task:

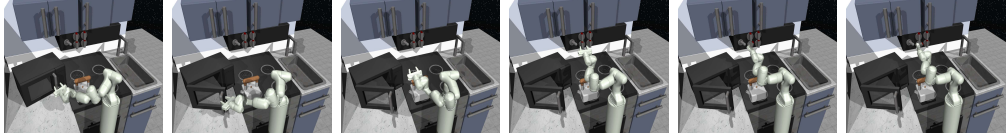


FIGURE 4.7: **Franka kitchen task visualization:** The visualization is a successful attempt at performing kitchen task.

In chapters 5 and 6, we leverage expert demonstrations for developing efficient hierarchical strategies and for solving complex long horizon tasks. Such demonstrations are near-optimal or user-required ways of solving the task, and are generated by either employing an expert human demonstrator or hard-coded demonstrator. We generate such expert demonstrations for the environments, and now provide the details of expert demonstrations generation.

4.2 Generating expert demonstrations

We explain the procedure for generating expert demonstrations as follows:

4.2.1 Maze navigation Environment

We use the path planning RRT algorithm to generate optimal paths $P = (p_t, p_{t+1}, p_{t+2}, \dots, p_n)$ from the initial state to the goal state. Using these expert paths, we generate state-action expert demonstration dataset for the lower level policy, which is later used to generate subgoal transition dataset. Since the procedure is automated using RRT algorithm, we can generate expert demonstrations without the expert.

4.2.2 Pick and place and Push Environment

For generating expert demonstrations, we initially used a human agent in virtual reality based Mujoco simulation to generate demonstrations. We later found that hard coding

a control policy also works reasonably well in this environment. Hence, we used a hard-coded policy to generate the expert demonstrations. In this task, the robot firstly picks up the block using robotic gripper, and then takes it to the target goal position.

4.2.3 Bin Environment

In this environment, we used a hard-coded policy to generate the expert demonstrations. In this task, the robot firstly picks up the block using robotic gripper, and then places it in the target bin. Using these expert trajectories, we generate expert demonstration dataset for the lower level policy.

4.2.4 Hollow Environment

In this environment, we used a hard-coded policy to generate the expert demonstrations. In this task, the robotic gripper has to pick up the square hollow block and place it such that a vertical structure on the table goes through the hollow block.

4.2.5 Rope Manipulation Environment

We hand coded an expert policy to automatically generate expert demonstrations $e = (s_0^e, s_1^e, \dots, s_{T-1}^e)$, where s_i^e are demonstration states. The states s_i^e here are rope configuration vectors. The expert policy is explained below.

Let the starting and goal rope configurations be sc and gc . We find the cylinder position pair (sc_m, gc_m) where $m \in [1, n]$, such that sc_m and gc_m are farthest from each other among all other cylinder pairs. Then, we perform a poke (x, y, θ) to drag sc_m towards gc_m . The (x, y) position of the poke is kept close to sc_m , and poke direction θ is the direction from sc_m towards gc_m . After the poke execution, the next pair of farthest cylinder pair is again selected and another poke is executed. This is repeatedly done for k pokes, until either the rope configuration sc comes within δ distance of goal gc , or we reach maximum episode horizon T . Although, this policy is not the perfect policy for goal

based rope manipulation, but it still is a good expert policy for collecting demonstrations D . Moreover, as our method requires states and not primitive actions (pokes), we can use these demonstrations D to collect good higher level subgoal dataset D_g using primitive parsing.

4.2.6 Kitchen Environment

In this environment, we used the expert demonstrations provided in D4RL dataset [90]. We use directed demonstrations from this dataset to solve the multi-stage task in this environment.

4.3 Real World Experiments

To evaluate whether the learned policies can be deployed in real-world robotic tasks, we implement pick-and-place, bin, and rope manipulation tasks using a Dobot Magician robot (Fig 4.8). This robot, being a relatively affordable robotic manipulator, was intentionally chosen to demonstrate that the proposed approaches can be effectively deployed on a simple setup for robotic navigation and manipulation tasks. We perform experiments on real pick and place, bin and rope manipulation tasks. We use Realsense D435 depth camera to extract the robotic arm position, block, bin, and rope cylinder positions. Computing accurate linear and angular velocities is hard in real tasks, so we assign them small hard-coded values, which shows good performance.

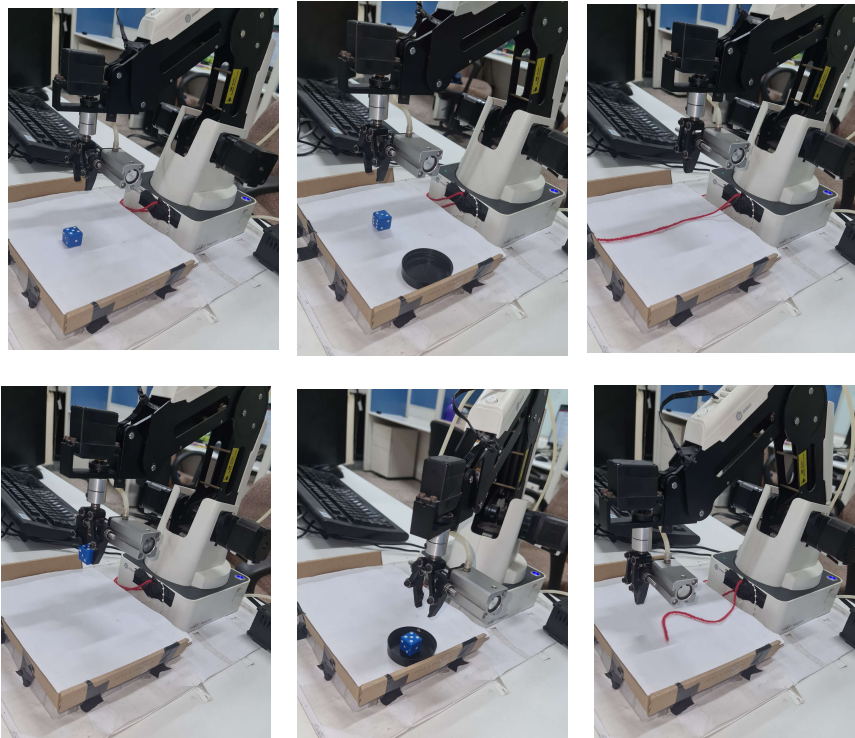


FIGURE 4.8: **Real world tasks** This figure depicts the three real world environments in our experiments: (a) Pick and Place (Col 1), (b) Bin (Col 2), and (c) Rope manipulation task (Col 3). In pick and place, the robotic gripper has to pick the block and bring it to goal position. In bin task, the gripper has to pick the block and place it in the bin. In kitchen task, the gripper has to poke to deformable rope, and align it to the goal rope configuration. Row 1 depicts initial state and Row 2 depicts final goal configurations.

Part II

Leveraging Expert Demonstrations

Chapter 5

CRISP: Curriculum-inducing Primitive Informed Subgoal Prediction

We learn by climbing from the simple to the complex.

– Anonymous

¹ Reinforcement learning (RL) algorithms have made significant progress in solving continuous control tasks like performing robotic arm manipulation [91, 92] and learning dexterous manipulation [93]. However, the success of RL algorithms on long horizon continuous tasks has been limited by issues like long term credit assignment and inefficient exploration [6, 94], especially in sparse reward scenarios [95]. Hierarchical reinforcement learning (HRL) [12, 96, 97] promises the benefits of temporal abstraction and efficient exploration [6] for solving tasks that require long term planning. In *goal-conditioned* hierarchical framework, the high-level policy predicts subgoals for lower primitive, which in

¹This chapter is based on the paper in submission at AAAI 2026 [82].

turn performs primitive actions directly on the environment [65–67]. However, simultaneously learning multi-level policies is challenging in practice due to non-stationary lower primitive behavior.

Prior works have leveraged expert demonstrations to bootstrap learning [93, 98, 99]. Some approaches rely on leveraging expert demonstrations to generate subgoals and subsequently bootstrapping multi-level hierarchical RL policies using behavior cloning [100]. In such approaches, generating an efficient subgoal transition dataset is crucial. Ideally, good subgoals should properly balance the task split between various hierarchical levels. Previous works like fixed parsing based approaches [100] may generate subgoals that are either too hard or too easy for the lower primitive, which leads to degenerate solutions. In this work, we propose an *adaptive parsing* technique to leverage expert demonstrations, that considers the current goal achieving ability of lower primitive when generating subgoals. CRISP predicts progressively harder subgoals to continuously evolving lower primitive, such that (i) the subgoals are always achievable by current lower primitive, (ii) the task split is balanced between hierarchical levels, and (iii) the policy eventually achieves the final goal. We employ these ideas and propose *curriculum inducing primitive informed subgoal prediction* (CRISP), a generally applicable HRL approach that introduces *hierarchical curriculum learning* to deal with non-stationarity.

Our approach is inspired from curriculum learning [101–108], where agent learns to solve the final task by solving increasing harder tasks. CRISP parses a handful of expert demonstrations using our novel subgoal relabeling approach: *primitive informed parsing* (PIP). In PIP, the current lower primitive is used to periodically perform data relabeling on expert demonstrations to generate efficient subgoal supervision for higher level policy. Notably, using the lower primitive to perform data relabeling effectively eliminates any explicit labeling or demonstration segmentation by an expert. The subgoal transition dataset generated by PIP is leveraged by higher policy using an additional inverse reinforcement learning (IRL) regularization objective, thus enabling curriculum based regularization. We devise an IRL regularizer [109–111] that constrains state marginal of learned policy to be close to that of expert demonstrations.

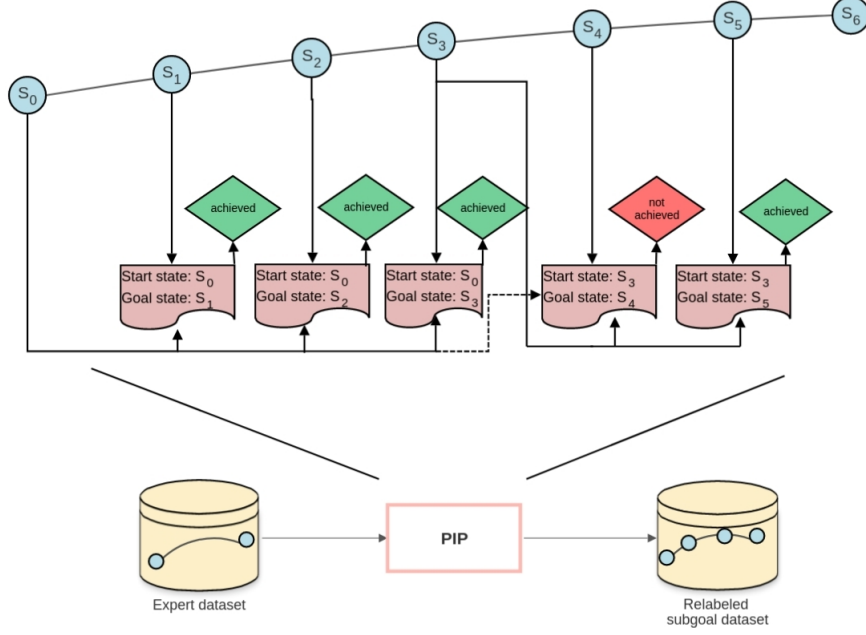


FIGURE 5.1: **PIP Overview:** PIP segments expert demonstrations by consecutively passing demonstration states as subgoals (s_i : for $i = 0$ to 6). If the lower primitive is unable to achieve state i in c timesteps (here $i = 4$), s_{i-1} (here s_3) is selected as subgoal for initial state s_0 , since s_{i-1} was the last reachable subgoal. The process is repeated with s_{i-1} (here s_3) as next initial state.

We perform experiments on complex maze navigation, pick and place, bin, hollow, rope manipulation and franka kitchen tasks, and empirically verify that CRISP improves sample efficiency and demonstrates impressive performance. We also perform real world experiments in robotic pick and place, bin and rope manipulation environments in Section 9.2, where CRISP shows impressive results. Please refer to Figure 5.1 for PIP overview and Figure 5.2 for subgoal curriculum generation. The main contributions of CRISP are as follows:

- CRISP proposes a novel primitive informed relabeling based approach to generate efficient subgoals, by employing the current lower primitive.
- Using IRL regularization, CRISP generates a curriculum of achievable subgoals for the lower primitive, thereby mitigating the non-stationarity issue in HRL.
- We empirically demonstrate that CRISP shows impressive performance in sparse robotic navigation and manipulation tasks, significantly outperforming the baselines.

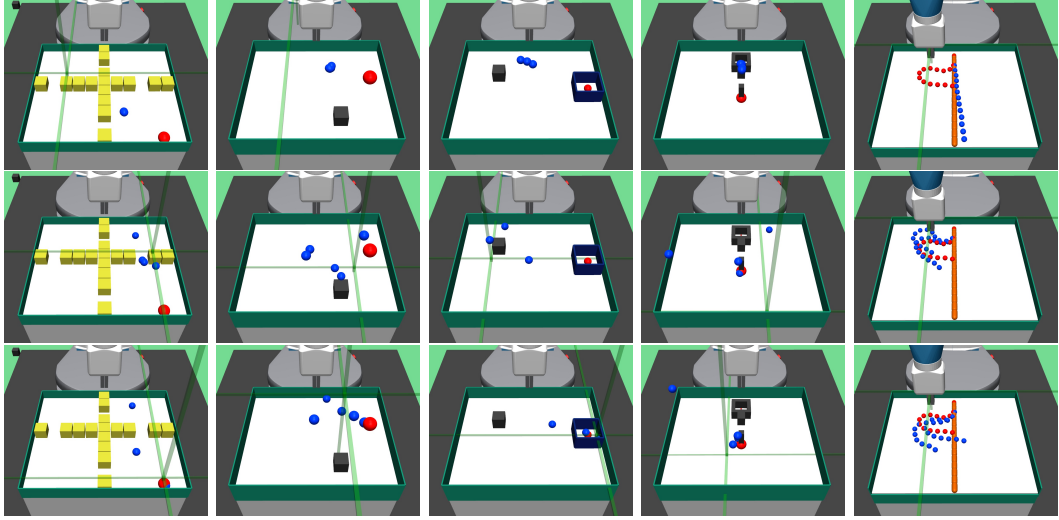


FIGURE 5.2: **Subgoal curriculum:** This figure depicts the progression of CRISP subgoal generation with training phases (Row 1: initial training, Row 2: mid training, Row 3: end training) in maze navigation (Col 1), pick and place (Col 2), bin (Col 3), hollow (Col 4), rope manipulation (Col 5) and kitchen environment (Col 6), (walls: yellow, final goal: red, subgoals: blue). CRISP produces a curriculum of reachable subgoals for lower primitive.

- We evaluate CRISP on various real world robotic tasks and show that CRISP exhibits impressive performance.

5.1 Curriculum-Inducing Subgoal Prediction

We consider *Universal Markov Decision Process* (UMDP) [112] setting, where Markov Decision processes (MDP) are augmented with the goal space G . UMDPs are represented as a 6-tuple (S, A, P, R, γ, G) , where S is the state space, A is the action space, $P(s'|s, a) = \mathbb{P}(s_{t+1} = s' | s_t = s, a_t = a)$ is the transition function that describes the probability of reaching state s' , when the agent takes action a in the current state s . The reward function R generates rewards r at every timestep. γ is the discount factor, and G is the goal space. In the UMDP setting, a fixed goal g is selected for an episode, and $\pi(a|s, g)$ denotes the goal-conditioned policy. The discounted future state distribution is represented as $d^\pi(s) = (1 - \gamma) \sum_{t=0}^T \gamma^t P(s_t = s | \pi)$, and the c -step future state distribution for policy π is represented as $d_c^\pi(s) = (1 - \gamma^c) \sum_{t=0}^T \gamma^{tc} P(s_{tc} = s | \pi)$. The overall objective is to

learn policy $\pi(a|s, g)$ which maximizes the expected future discounted reward objective $J = (1 - \gamma)^{-1} \mathbb{E}_{s \sim d^\pi, a \sim \pi(a|s, g), g \sim G} [r(s_t, a_t, g)]$

Let s be the current state and g be the final goal for the current episode. In our goal-conditioned hierarchical RL setup, the overall policy π is divided into multi-level policies. The higher level policy $\pi^H(s_g|s, g)$ predicts subgoals [96] s_g for the lower level primitive $\pi^L(a|s, s_g)$, which in turn executes primitive actions a directly on the environment. The lower primitive π^L tries to achieve subgoal s_g within c timesteps, by maximizing intrinsic rewards r_{in} provided by the higher level policy. The higher level policy π^H gets extrinsic reward r_{ex} from the environment, and predicts the next subgoal s_g for the lower primitive. The process is continued until either the final goal g is achieved, or the episode terminates. We consider sparse reward setting where the lower primitive is sparsely rewarded intrinsic reward 0 if the agent reaches within δ^L distance of the predicted subgoal s_g and -1 otherwise: $r_{in} = -1(\|s_t - s_g\|_2 > \delta^L)$, and the higher level policy is sparsely rewarded extrinsic reward 0 if the achieved goal is within δ^H distance of the final goal g , and -1 otherwise: $r_{ex} = -1(\|s_t - g\|_2 > \delta^H)$. The expert demonstrations are represented as $D = \{e^i\}_{i=1}^N$, where $e^i = (s_0^e, s_1^e, \dots, s_{T-1}^e)$.

We first explain our primitive informed parsing scheme (PIP) for selecting efficient subgoals according to the current lower primitive, and then explain our hierarchical curriculum learning based approach that mitigates non-stationarity in HRL. PIP periodically performs data relabeling on expert demonstrations to populate subgoal transition dataset D_g , which is used subsequently in IRL based regularization objective.

5.1.1 Primitive Informed Parsing: PIP

Primitive informed parsing approach (PIP) uses the current lower primitive π_L to parse expert state demonstrations dataset D . An overview of the method is depicted in Figure 5.1. We explain how PIP adaptively parses expert demonstration trajectories from D to generate subgoal transition dataset D_g . We start with current lower primitive π_L and an expert state demonstration trajectory $e = (s_0^e, s_1^e, \dots, s_{T-1}^e)$. The environment is

reset to initial state s_0^e . Starting at $i = 1$ to $T - 1$, we incrementally provide states s_i^e as subgoals to lower primitive π_L . π_L tries to achieve s_i^e within c timesteps from the initial state. If π_L achieves the subgoal s_i^e within c timesteps, we provide s_{i+1}^e as the next subgoal. Conversely, if π_L fails to achieve the subgoal s_i^e from the initial state, we add s_{i-1}^e to the list of subgoals. Intuitively, since s_{i-1}^e was the last subgoal achieved by lower-level primitive, it is a good candidate for the next subgoal from initial state. Once we add s_{i-1}^e to the list of subgoals, we continue the process after setting s_{i-1}^e as the new initial state, until we reach the end of e . This subgoal transition sequence D_g^e is subsequently added to D_g . The PIP pseudocode is given in Algorithm 1.

Notably, PIP assumes ability to reset the environment to any state in D . Although this seems impracticable in real world robotic scenarios, this becomes feasible in our setup since we first learn good policies in simulation, and then deploy them in real world robotic scenarios. The underlying assumption is that with enough training in simulation, the policy becomes general enough to perform well in real world tasks. We perform extensive experiments to support this claim in Section 5.2 and discuss various ways to relax this assumption in Section 5.4.

Algorithm 1 PIP: Primitive Informed Parsing

```

Initialize  $D_g = \{\}$ 
(final goal)  $s_f = g$ 
for each trajectory  $e = (s_0^e, s_1^e, \dots, s_{T-1}^e)$  in  $D$  do
  (initial state)  $s_{in} \leftarrow s_0^e$ 
  Initialize list of subgoals  $D_g^e = \{\}$ 
  for  $i = 1$  to  $T - 1$  do
    Reset to initial state  $s_{in}$ 
    Pass  $s_i^e$  as the current goal to  $\pi_L$ 
    if  $s_i^e$  is not achieved by  $\pi_L$  in  $c$  time-steps then
      Add  $(s_{in}, s_{i-1}^e, s_f)$  to  $D_g^e$ 
       $s_{in} \leftarrow s_{i-1}^e$ 
   $D_g \leftarrow D_g \cup D_g^e$ 

```

5.1.2 Inverse RL regularization

Here, we explain CRISP uses D_g , the subgoal transition dataset generated using PIP to learn an IRL regularizer, which regularizes the higher level policy into predicting a curriculum of achievable goals for the lower primitive. We devise IRL objective as a GAIL [111] like objective implemented using LSGAN [113]. Let $(s^e, s_g^e, s_{next}^e) \sim D_g$ be higher level subgoal transition from an expert trajectory where s^e is current state, s_{next}^e is next state, g^e is final goal and s_g^e is subgoal supervision. Let s_g be the subgoal predicted by the high level policy $\pi_{\theta_H}^H(\cdot | s^e, g^e)$ with parameters θ_H , and $\mathbb{D}_{\epsilon_H}^H$ be the higher level discriminator with parameters ϵ_H . J_D^H represents upper level IRL objective, which depends on parameters (θ_H, ϵ_H) . We bootstrap the learning of higher level policy by optimizing:

$$\begin{aligned} \max_{\theta_H} \min_{\epsilon_H} J_D^H(\theta_H, \epsilon_H) = \max_{\theta_H} \min_{\epsilon_H} & \frac{1}{2} \mathbb{E}_{(s^e, s_g^e, \cdot) \sim D_g} [\mathbb{D}_{\epsilon_H}^H(s_g^e) - 1]^2 \\ & + \frac{1}{2} \mathbb{E}_{(s^e, \cdot, \cdot) \sim D_g, s_g \sim \pi_{\theta_H}^H(\cdot | s^e, g^e)} [\mathbb{D}_{\epsilon_H}^H(\pi_{\theta_H}^H(\cdot | s^e, g^e)) - 0]^2. \end{aligned} \quad (5.1)$$

This objective forces the higher policy subgoal predictions to be close to subgoal predictions of the dataset D_g . The discriminator $\mathbb{D}_{\epsilon_H}^H$ creates a natural curriculum for regularizing higher level policy by assigning the value 1 to the predicted subgoals that are closer to the subgoals from dataset D_g , and 0 otherwise. $\mathbb{D}_{\epsilon_H}^H$ improves with training, and regularizes the higher policy to predict achievable subgoals for the lower primitive.

Similarly for lower level primitive, let $(s^f, a^f, s_{next}^f) \sim D_g^L$ be lower level expert transition where s^f is current state, s_{next}^f is next state, g^f is final goal, a is the primitive action predicted by lower policy $\pi_{\theta_L}^L(\cdot | s^f, s_g^e)$ with parameters θ_L , and $\mathbb{D}_{\epsilon_L}^L$ be the lower level discriminator with parameters ϵ_L . Let J_D^L represent lower level IRL objective, which depends on parameters (θ_L, ϵ_L) . The lower level IRL objective is thus:

$$\begin{aligned} \max_{\theta_L} \min_{\epsilon_L} J_D^L(\theta_L, \epsilon_L) = \max_{\theta_L} \min_{\epsilon_L} & \frac{1}{2} \mathbb{E}_{(s^f, a^f, \cdot) \sim D_g^L} [\mathbb{D}_{\epsilon_L}^L(a^f) - 1]^2 \\ & + \frac{1}{2} \mathbb{E}_{(s^f, \cdot, \cdot) \sim D_g^L, a \sim \pi_{\theta_L}^L(\cdot | s^f, s_g^e)} [\mathbb{D}_{\epsilon_L}^L(\pi_{\theta_L}^L(\cdot | s^f, s_g^e)) - 0]^2. \end{aligned} \quad (5.2)$$

5.1.3 Joint optimization

Finally, the higher level policy is trained to produce subgoals, which when fed into the lower level primitive, maximize the sum of future discounted rewards for our task using off-policy reinforcement learning. Here T is the task horizon and g is the sampled goal for the current episode. For brevity, we can refer to this objective function as $J_{\theta_H}^H$ and $J_{\theta_L}^L$ for upper and lower level policies. We use the IRL objective to regularize the high level off-policy RL objective:

$$\max_{\theta_H} (J_{\theta_H}^H + \psi * (\min_{\epsilon_H} J_D^H(\theta_H, \epsilon_H))), \quad (5.3)$$

whereas the lower level primitive is trained by optimizing:

$$\max_{\theta_L} (J_{\theta_L}^L + \psi * (\min_{\epsilon_L} J_D^L(\theta_L, \epsilon_L))). \quad (5.4)$$

The lower policy is regularized using expert demonstration dataset, and the upper level is optimized using subgoal transition dataset populated using PIP. ψ is the regularization weight for the IRL objective. When $\psi = 0$, the method reduces to HRL policy with no higher level policy regularization, and large values of ψ may lead to overfitting to the expert demonstrations. Refer to Algorithm 2 for CRISP algorithm.

Algorithm 2 CRISP

Require: D (expert demonstrations)
 p (population hyperparameter)
Initialize higher level subgoal transition dataset $D_g = \{\}$
for epoch $i = 1 \dots N$ **do**
 if $i \% p == 0$ **then**
 Clear D_g
 Populate D_g by relabeling D using PIP
 for $j = 1$ **to** $T - 1$ **do**
 Collect experience using π_H and π_L
 Update lower policy via RL and IRL (Eq 6.8)
 Sample transitions from D_g
 Update higher policy via RL and IRL (Eq 6.7)

5.2 Experiments

In this section, we perform experimental analysis to answer the following questions:

1. Does the adaptive relabeling approach CRISP outperform fixed window parsing based approach?
2. How effectively does CRISP improve sample efficiency and stabilize HRL training compared to other baselines?
3. Is CRISP able to mitigate non-stationarity, thereby stabilizing HRL training?
4. Does the integration of IRL regularization contribute to generating a curriculum of achievable subgoals?
5. How well does CRISP perform when deployed in real-world robotic environments?
6. What are the qualitative insights into CRISP’s efficacy over baselines?
7. What is the individual impact of each design component within CRISP?

Environment and Implementation details We perform experiments on six complex robotic environments with continuous state and action spaces that require long term planning: (i) maze navigation, (ii) pick and place, (iii) bin, (iv) hollow, (v) rope manipulation, and (vi) franka kitchen. We use the off-policy Soft Actor Critic [114] as RL algorithm with Adam [115] optimizer.

5.2.1 Additional hyper-parameters

Here, we enlist the additional hyper-parameters used in CRISP:

Hyperparameter	Value	Description
activation	tanh	Activation for hierarchical policies
layers	3	Number of layers in the critic/actor networks
hidden	512	Number of neurons in each hidden layer
Q_lr	0.001	Critic learning rate
pi_lr	0.001	Actor learning rate
buffer_size	int(1E7)	Size of experience replay buffer
tau	0.8	Polyak averaging coefficient
clip_obs	200	Clip observation
n_cycles	1	Number of cycles per epoch
n_batches	10	Training batches per cycle
batch_size	1024	Batch size hyper-parameter
random_eps	0.2	Percentage of time a random action is taken
alpha	0.05	Weightage parameter for SAC
noise_eps	0.05	Std of Gaussian noise added to not-completely-random actions
norm_eps	0.01	Epsilon used for observation normalization
norm_clip	5	Normalized observations are cropped to this value
adam_beta1	0.9	Beta 1 for Adam optimizer
adam_beta2	0.999	Beta 2 for Adam optimizer

TABLE 5.1: Hyperparameters for the model setup.

5.2.2 Environment and Implementation details

We collect 28 expert demonstrations in kitchen and 100 demonstrations in all other tasks. For collecting expert demonstrations, we use RRT based trajectories in maze task, Mujoco VR based direction in pick and place, bin and hollow tasks, poking based expert controller in rope task, and D4RL expert data in kitchen task.

5.2.3 RAPS baseline setup

The hand designed action primitives from RAPS are designed as follows: (i) in maze navigation, the lower level primitive travels in a straight line directly towards the subgoal predicted by higher level policy, (ii) in pick and place, bin and hollow tasks, we hand-designed three primitives: *gripper-reach* (where the gripper has to reach the goal position, *gripper-open* (where the robotic arm has to open the gripper) and *gripper-close* (where the robotic arm has to close the gripper). In kitchen environment, we use the action primitives implemented in RAPS. Since it is hard to design the action primitives in rope environment, we do not evaluate RAPS in the rope environment. We use off-policy Soft Actor Critic [114] as RL algorithm with Adam [115] optimizer. Unless otherwise stated, we keep the training conditions consistent across all baselines to ascertain fair comparisons, and empirically tune the hyper-parameter values of our method and all other baselines.

Sparse Reward and Goal-Conditioning Complexity. While some tasks may appear simple, all evaluation environments employ *sparse rewards*, where the agent has to extensively explore the environment before coming across any rewards, thus significantly increasing task complexity. For example, the Franka Kitchen task only rewards completion of the last subtask (opening the microwave and turning on the gas knob). Additionally, we consider goal-conditioned RL setting, where the final goals are randomly generated, which also increases the task complexity. These design choices elevate task complexity and highlight the advantages of learning efficient hierarchical policies with curriculum generation, as naive or flat baseline methods struggle to make progress.

Evaluating two CRISP regularizers: Along with our RL objective, We evaluate CRISP using two imitation learning regularizers: IRL regularization (CRISP-IRL) and BC regularization (CRISP-BC). Evaluating both BC and IRL variants offers complementary insights due to their fundamentally different approaches to imitation learning (IL). BC directly learns a mapping from states to expert actions, making it sample-efficient and effective when demonstrations are abundant, high-quality, and cover the relevant state distribution. However, BC is prone to compounding errors and struggles with distributional shift, which can degrade performance in complex or noisy environments. IRL, in contrast, infers the underlying reward function guiding the expert’s behavior, enabling the agent to generalize beyond demonstrations by optimizing policies that reflect the expert’s intent—even in unseen states. This often leads to better robustness and the ability to surpass imperfect experts but can require more complex training. By separately implementing both IRL and BC variants, we gain a clearer understanding of their relative strengths and weaknesses across different tasks, and also renders insights into which IL approach is better suited for specific robotic control tasks.

Figure 5.4, we depict the success rate performance of CRISP and compare it with other baselines. The results are averaged over 5 randomly selected seeds. Unless otherwise stated, we keep the training conditions consistent across all baselines to ascertain fair comparisons, and empirically tune the hyper-parameter values of our method and all other baselines.

1. Does our adaptive relabeling based approach CRISP outperform fixed window parsing based approaches?

In Figure 5.4, we compare CRISP with *RPL* (Relay Policy Learning) [100], to demonstrate the efficacy of adaptive relabeling compared to fixed window based relabeling. RPL first uses supervised pre-training from undirected demonstrations, and then fine-tunes the policy using RL. To ensure fair comparisons, we use a variant of *RPL* which does not use this pre-training. Thus, the only difference between CRISP and RPL is that CRISP uses adaptive relabeling to select subgoals, and RPL uses fixed window based relabeling.

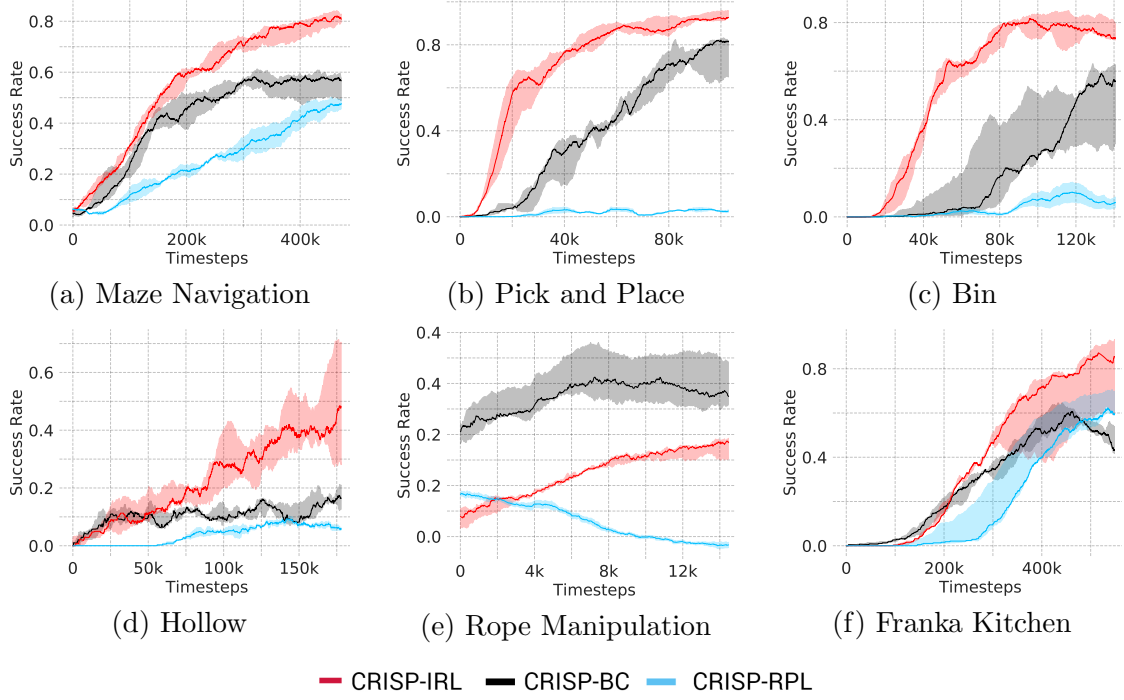


FIGURE 5.3: **CRISP-RPL ablation** This figure depicts comparison between CRISP-IRL, CRISP-BC, and CRISP-RPL ablation on six sparse robotic environments: Maze Navigation (a), Pick and Place (b), Bin (c), Hollow (d), Rope Manipulation (e), and Franka Kitchen environment (f).

As illustrated in Figure 5.4, CRISP consistently outperforms *RPL* across all evaluated tasks, highlighting the effectiveness of adaptive relabeling over fixed window-based approaches. While both methods leverage a combination of reinforcement learning and imitation learning, their key difference is in how subgoals are assigned. CRISP’s adaptive relabeling dynamically selects subgoals that are achievable by the current lower-level primitive, thus automatically adjusting the subgoal difficulty without the need for manual tuning.

This adaptability proves especially valuable when perturbations or random exploration cause the agent to deviate from the demonstration trajectory. In such cases, CRISP’s high-level policy responds by generating future subgoals that steer the primitive back toward success, enabling recovery from off-manifold states where RPL typically fails. For example, in rope manipulation experiments, agents using fixed window relabeling often become stuck after a poor poke, unable to recover. In contrast, CRISP promptly proposes

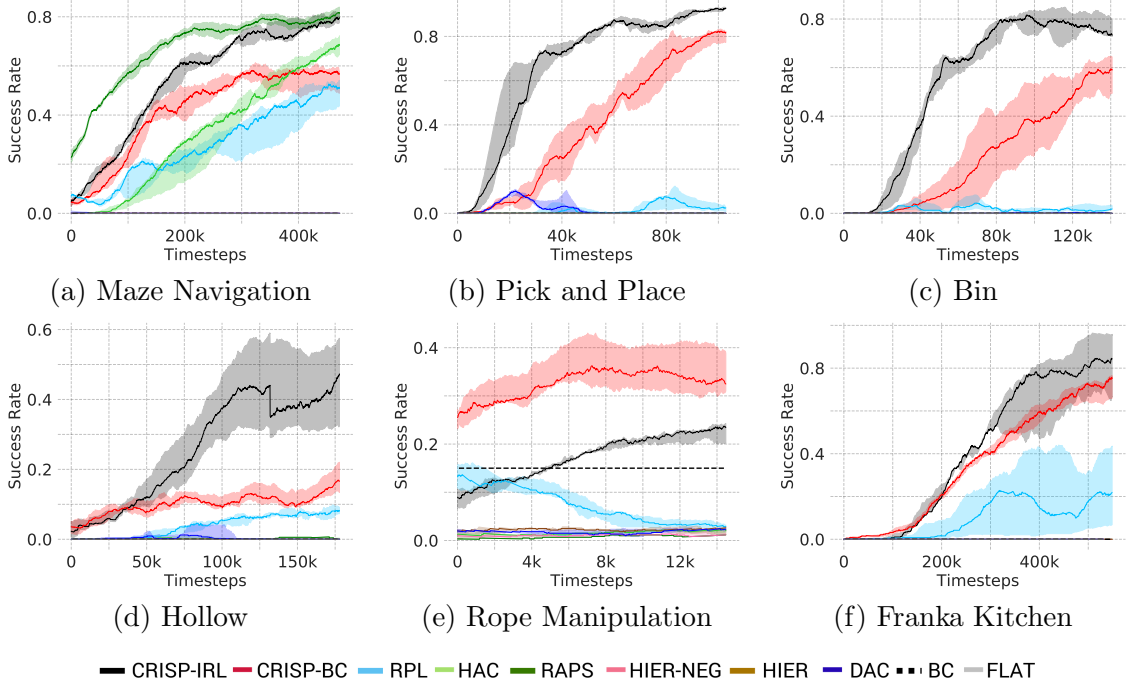


FIGURE 5.4: **Success rate comparison** This figure compares the success rate performances on six sparse robotic environments: Maze Navigation (a), Pick and Place (b), Bin (c), Hollow (d), Rope Manipulation (e), and Franka Kitchen environment (f). The solid line represents the mean and shaded region represents the range of success rates across 5 seeds. We compare our approaches CRISP-IRL and CRISP-BC against multiple baselines: RPL (Relay Policy learning [100]), HAC (Hierarchical Actor Critic [67], RAPS [116], HIER-NEG (HRL implementation using SAC where higher policy is negatively rewarded when lower primitive fails to achieve the predicted subgoal), HIER (HRL implementation using SAC), DAC [110], and FLAT (Single-level RL policy using SAC). As seen from the plots, CRISP shows impressive performance and significantly outperforms the baselines.

a reachable intermediate rope configuration, allowing the agent to proceed and ultimately succeed.

We also elucidate the importance of adaptive relabeling by considering a variant of CRISP with the subgoal dataset collected using fixed window based relabeling. We compare this variant (*CRISP-RPL*) with our approach in Figure 5.5. CRISP consistently outperforms this baseline, showing that adaptive relabeling is crucial for improved performance.

2. Does CRISP improve sample efficiency and stabilize HRL training compared to other baselines?

We evaluate CRISP against a broad spectrum of hierarchical and non-hierarchical baselines to assess its impact on sample efficiency and stability during training. The Hierarchical

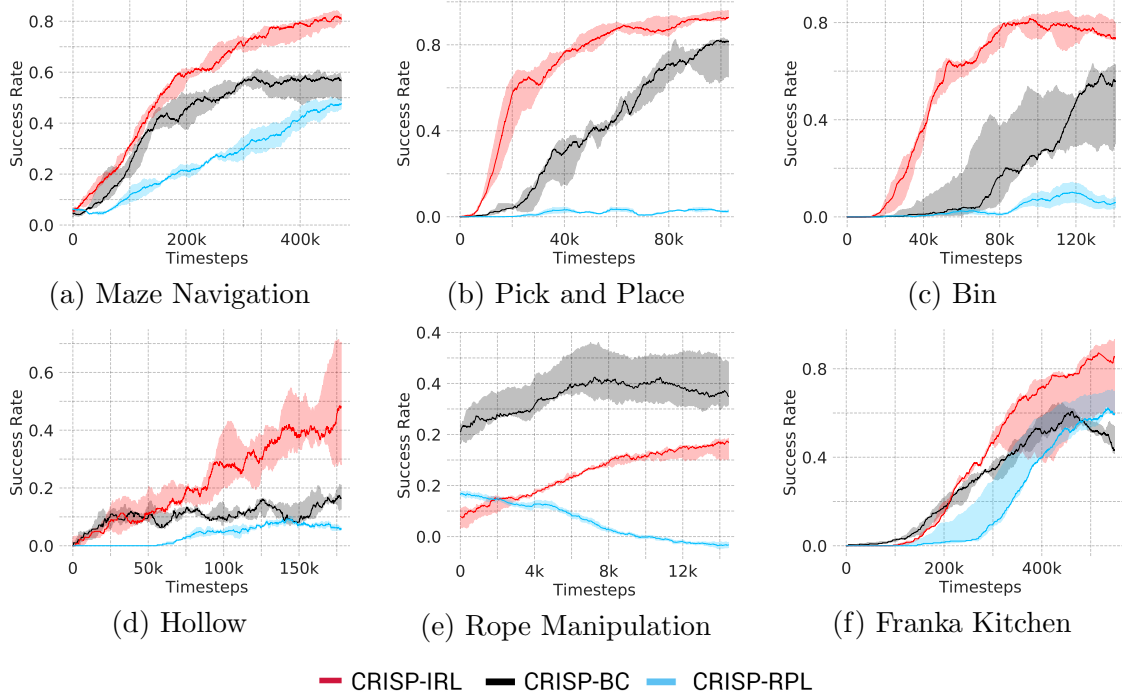


FIGURE 5.5: **CRISP-RPL ablation** This figure depicts comparison between CRISP-IRL, CRISP-BC, and CRISP-RPL ablation on six sparse robotic environments: Maze Navigation (a), Pick and Place (b), Bin (c), Hollow (d), Rope Manipulation (e), and Franka Kitchen environment (f).

Actor Critic (HAC) [67] is a hierarchical approach to generate subgoals by assuming the lower-level policy is optimal. Although this mitigates non-stationarity, this assumption does not hold in practice when the hierarchical levels are learned simultaneously, which may produce sub-optimal subgoals. As illustrated in Figure 5.4, although HAC achieves competitive performance on simpler maze navigation task, it struggles in harder tasks. In contrast, CRISP’s adaptive subgoal relabeling dynamically aligns the subgoal difficulty with the evolving capabilities of the lower-level policy, enabling more stable hierarchical training, achievable subgoal generation and leading to superior sample efficiency and higher success rates across tasks.

In recent works like *RAPS* [116], hand-designed action primitives are used to predict primitive actions. In such scenarios, the goal of the upper level is to pick the optimal sequence of action primitives for solving the task. However, such approaches require considerable effort in hand designing such primitives. Except maze navigation task, *RAPS*

is unable to perform well in other environments. We believe that this is because in such navigation tasks, the task of the higher level is reduced to predicting the correct direction for the agent to move. However, in other tasks, predicting the primitives for lower primitive is hard. Notably, we re-implemented the environments to be sparsely rewarded, which accounts for poor performance of *RAPS*. In contrast, CRISP shows impressive results and is able to outperform RAPS in these complex manipulation tasks, without requiring access to such domain-specific action primitives. Notably, we do not evaluate RAPS in rope task for this work, since designing the RAPS action primitives is hard in this environment.

We also compare our approach with two other hierarchical baselines: *HIER* and *HIER-NEG*, which are hierarchical off-policy SAC based baselines that do not leverage expert demonstrations. In *HIER-NEG*, the higher level policy is negatively rewarded if the lower primitive is unable to reach the predicted subgoal. CRISP achieves notably higher performance compared to these hierarchical baselines, indicating that improvements stem not only from hierarchical abstraction but from our novel adaptive relabeling and primitive regularization, that directly mitigates non-stationarity and enforces subgoal feasibility.

Finally, we compare CRISP against flat baselines, including Discriminator Actor Critic (DAC), single-level RL (FLAT), and behavior cloning (BC). These baselines fail to reliably solve the tasks tested, reinforcing the critical importance of both hierarchical abstraction and curriculum-based subgoal adaptation. Thus, our comprehensive evaluation demonstrates that CRISP significantly improves sample efficiency and stabilizes learning compared to existing hierarchical and flat baselines.

3. Is CRISP able to mitigate non-stationarity, thereby stabilizing HRL training?

We illustrate CRISP’s ability to mitigate non-stationarity in HRL in Figure 5.6. To quantify this, we compare CRISP to the *HIER* baseline by measuring the average distance during training between the subgoals issued by the higher-level policy and the actual states

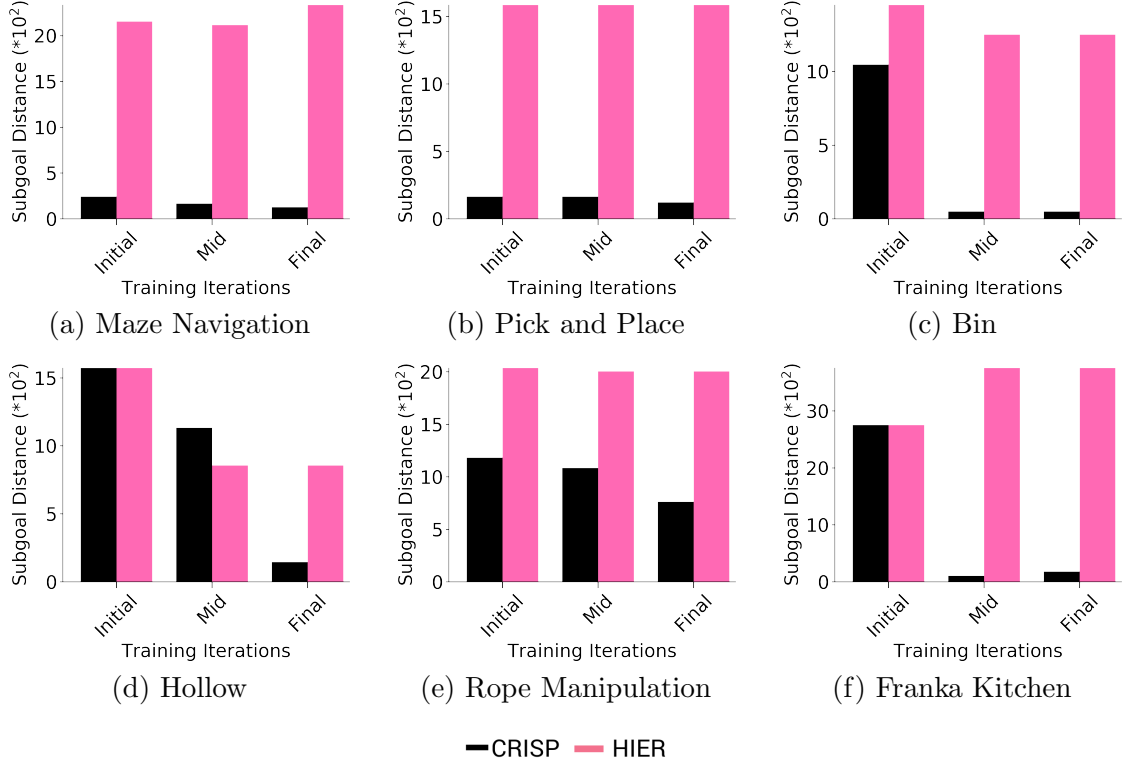


FIGURE 5.6: **Non-stationarity metric comparison** This figure compares the average distance metric between the subgoals predicted by the higher level policy and the states achieved by the lower level policy during training on six sparse robotic environments: Maze Navigation (a), Pick and Place (b), Bin (c), Hollow (d), Rope Manipulation (e), and Franka Kitchen environment (f). As seen in the figure, CRISP consistently produces efficient subgoals leading to low distances between the predicted and achieved subgoals throughout the training process. This mitigates non-stationarity in HRL.

reached by the lower-level policy. Lower distance values indicate that the higher-level policy is generating subgoals that are well-aligned with the capabilities of the lower-level policy, which leads to more consistent goal achievement and reduces non-stationarity within the hierarchical learning process. As shown, CRISP consistently produces achievable subgoals throughout training, resulting in superior stability and effective non-stationarity mitigation.

4. Does our PIP based IRL regularization contribute to generating a curriculum of achievable subgoals?

Primitive-Informed Parsing (PIP) and IRL regularization plays a crucial role in generating a curriculum of subgoals that dynamically adapt to the low-level policy’s evolving

capabilities. PIP segments expert demonstrations by identifying subgoals that the current low-level controller can realistically achieve within a given horizon, effectively ensuring that each subgoal is feasible. Subsequently, IRL regularization regularizes the higher level policy to predict such achievable subgoals according to the current lower level primitive. This adaptive parsing leads to an implicit curriculum: early in training, subgoals are simpler and localized, while later stages naturally involve more complex and distant targets as the primitive improves. This is evident in Figure 5.2, where early in training CRISP proposes simpler, nearby subgoals (Row 1), and as training advances (Rows 2 and 3), it progressively generates more challenging and complex subgoals that effectively facilitate progress towards the final objective.

5. How well does CRISP perform when deployed in real-world robotic environments?

In order to analyse whether the learnt policies can be deployed in real world robotic tasks, we perform experiments on real pick and place, bin and rope manipulation tasks. We use Realsense D435 depth camera to extract the robotic arm position, block, bin, and rope cylinder positions. Since accurately computing accurate linear and angular velocities is hard in real tasks, we assign them small values, which we found to be a reasonable assumption in practice. We performed 5 sets of experiments with 10 trial each, and report the average success rates.

In real-robot experiments (Fig. 4.8), the ability to structure the task into feasible subgoals is essential: both CRISP-IRL and CRISP-BC successfully decompose complex tasks into reachable stages, leading to notably higher real-world success rates. Further, by ensuring that subgoals correspond to attainable states, these methods inherently promote safer operation, as the robot avoids attempting unsafe or unreachable maneuvers. CRISP-IRL achieves an accuracy of 0.6, 0.6 and 0.1, whereas CRISP-BC achieves accuracy of 0.8, 0.3, 0.2 on pick and place, bin and rope tasks respectively. We also deployed the next best performing baseline *RPL* on the three tasks, however since it is unable to generate good subgoals, the policy gets stuck and it fails to show any progress in the tasks.

6. What are the qualitative insights into CRISP’s efficacy over baselines?

We provide qualitative insights that clarify how our hierarchical framework distinctly leverages subgoals and hierarchy to outperform competing deep RL approaches. These insights complement quantitative metrics by illustrating the core mechanisms driving improved performance.

- 1. Adaptive Subgoal Curriculum:** Unlike baselines that rely on fixed or heuristic subgoal segmentation, our method dynamically generates subgoals tailored to the *current* ability of the low-level policy. Early in training, subgoals are simple and incrementally spaced, ensuring reachability and enabling steady progress. As the low-level agent improves, subgoals naturally become more challenging and spaced further apart, effectively creating an automatic curriculum that guides learning. This avoids common failure modes of rigid or fixed hierarchies where unreachable subgoals stall learning.
- 2. Hierarchical Decomposition:** Our high-level policy decomposes complex tasks into interpretable and achievable sub-tasks, such as “approach object,” “grasp object,” and “move to target location.” This explicit decomposition is absent in non-hierarchical or flat RL baselines that often treat the task monolithically. By breaking down long-horizon tasks into manageable chunks, our approach not only improves sample efficiency but also enables smoother and more stable learning trajectories.
- 3. Mitigation of Non-Stationarity:** By periodically re-constructing the subgoal transition dataset based on the evolving low-level policy performance, our approach ensures that high-level subgoals remain feasible throughout training. This technique directly mitigates notorious non-stationarity issues in hierarchical RL, leading to more stable learning and improved convergence compared to alternative approaches which issue outdated or infeasible subgoals.
- 4. Concrete Behavioral Differences from Competing Methods:** In tasks such as robotic rope manipulation, baseline methods often produce abrupt or globally

misguided actions, failing to make incremental progress. Our method, by contrast, enables the agent to complete the task through a sequence of reachable intermediate rope configurations, reflecting an effective hierarchical planner. This results in visibly smoother and more purposeful agent behavior.

Summary These qualitative characteristics demonstrate that our hierarchical method addresses fundamental challenges in hierarchical RL—such as non-stationarity and subgoal feasibility—through a principled and adaptive curriculum. This yields superior, stable performance and interpretable learning behaviors that competitor methods, relying on fixed hierarchies or flat policies, do not exhibit.

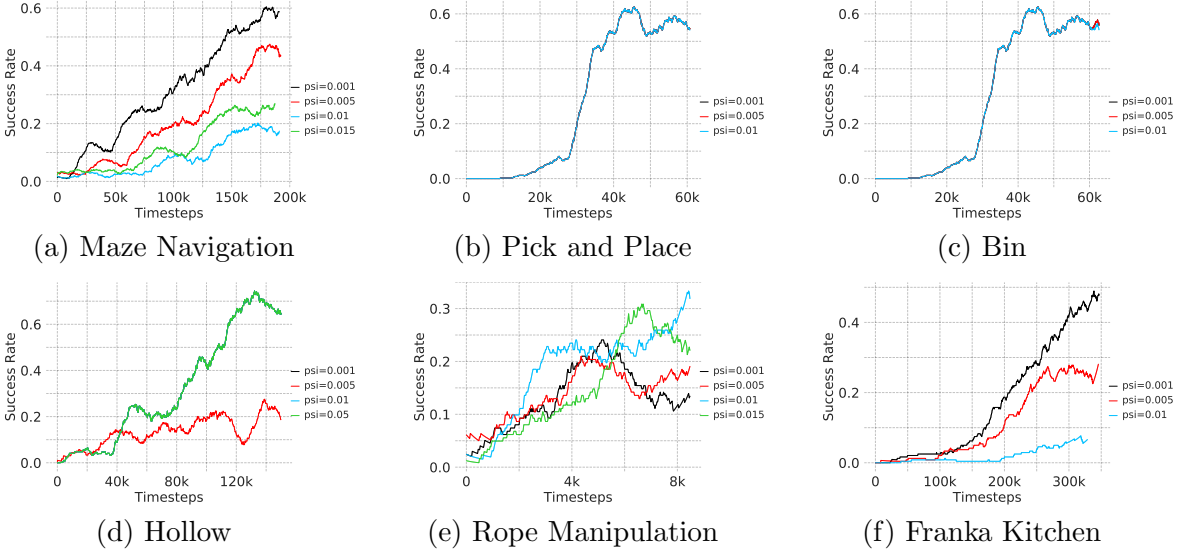


FIGURE 5.7: ψ **ablation** This figure depicts comparisons for various learning rate ψ values on six sparse robotic environments: Maze Navigation (a), Pick and Place (b), Bin (c), Hollow (d), Rope Manipulation (e), and Franka Kitchen environment (f).

7. What is the individual impact of each design component within CRISP?

We also perform ablations to select the (i) learning rate ψ (Figure 5.7), and population hyperparameter p (Figure 5.8). We empirically found that very large values of p are unable to generate good curriculum of subgoals. Further, when ψ is too high, the method might overfit to expert data, whereas if ψ is too small, CRISP is unable to utilize IL regularization.

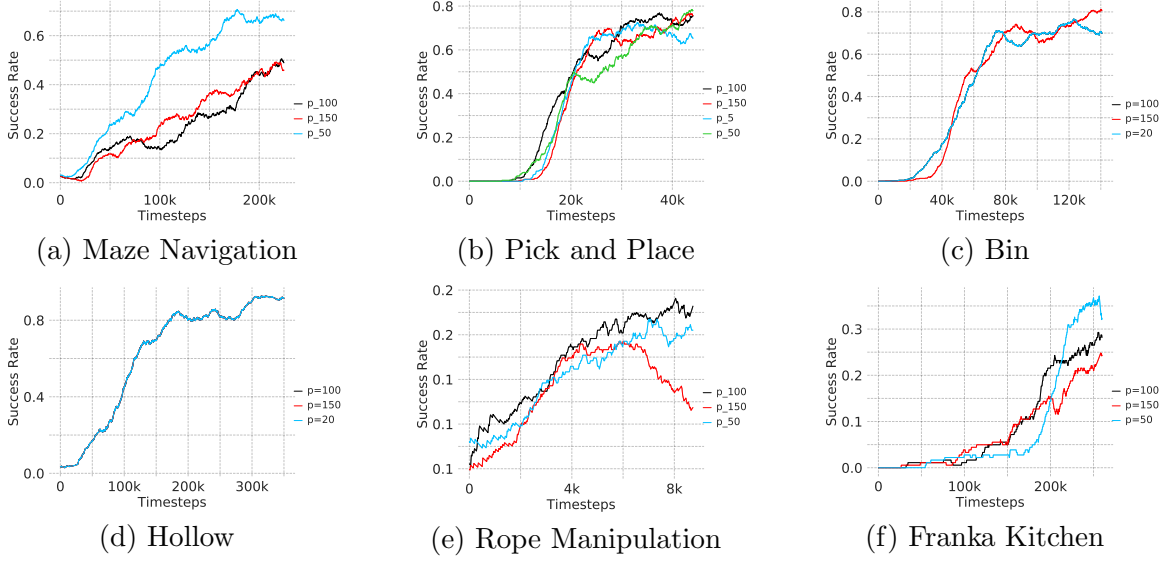


FIGURE 5.8: **p ablation** This figure depicts comparisons for various p values on six sparse robotic environments: Maze Navigation (a), Pick and Place (b), Bin (c), Hollow (d), Rope Manipulation (e), and Franka Kitchen environment (f).

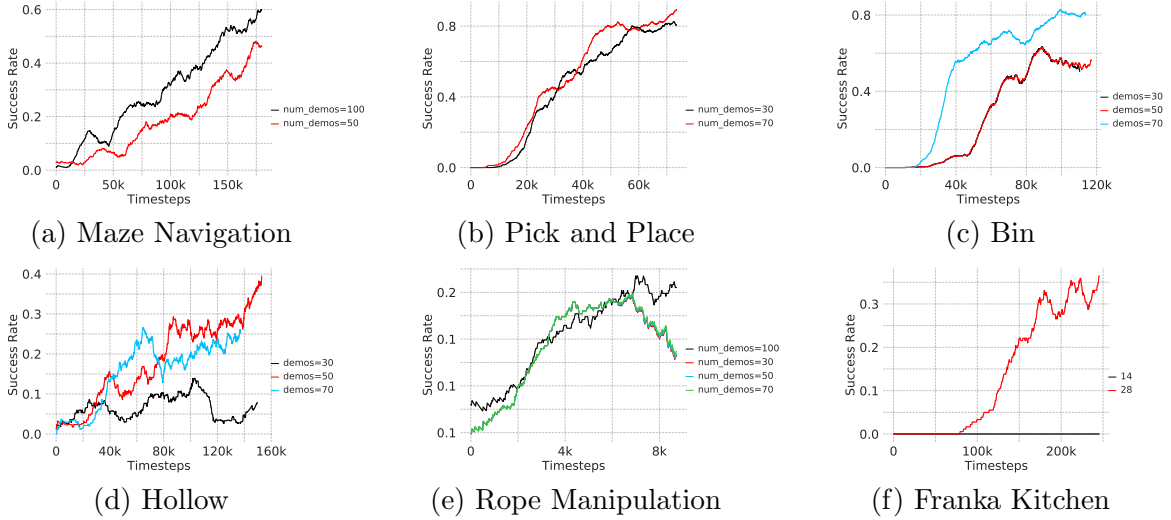


FIGURE 5.9: **Number of expert demonstrations ablation** This figure depicts ablation experiments of varying number of expert demonstrations versus number of training epochs on six sparse robotic environments: Maze Navigation (a), Pick and Place (b), Bin (c), Hollow (d), Rope Manipulation (e), and Franka Kitchen environment (f).

We also perform ablations to deduce the optimal number of expert demos required for each task (Figure 5.9). If the number of expert demos is too small, the policy may overfit. Although the number of expert demos are subject to availability, we increase the number until there is no significant improvement in performance. We also analyse the effect of varying the quality of expert data, and found direct co-relation quality of expert data and

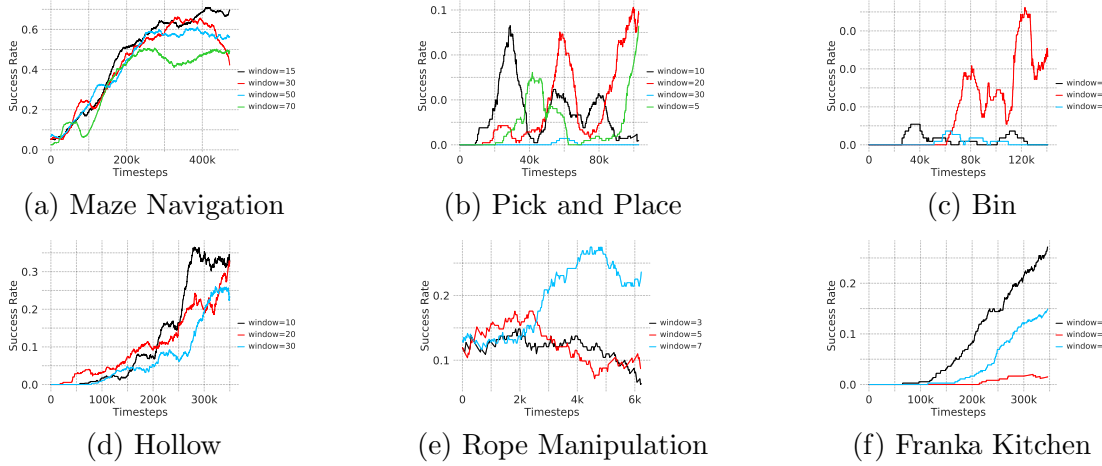


FIGURE 5.10: **Window size ablation** This figure depicts ablation experiments for the window size hyper-parameter in RPL on six sparse robotic environments: Maze Navigation (a), Pick and Place (b), Bin (c), Hollow (d), Rope Manipulation (e), and Franka Kitchen environment (f).

performance. We empirically choose the best value of window size hyper-parameter k on RP experiments (Figure 5.10).

5.3 Related Work

Learning from demonstrations Previous approaches [93, 98, 99] that leverage expert demonstrations to solve complex tasks have demonstrated impressive results. Expert demonstrations have been used to bootstrap option learning [117–120]. Other approaches use imitation learning to bootstrap hierarchical approaches in complex task domains [71, 120–122]. Prior work [100] uses fixed window based approach for parsing expert demonstrations to generate subgoal transition dataset for training higher level policy. However, such methods might produce sub-optimal subgoals for the lower primitive. In contrast, our relabeling approach (PIP) segments expert demonstration trajectories into *meaningful* subtasks, thus balancing the task-split between hierarchical levels. Additionally, our approach considers the limited goal achieving capability of lower primitive for subgoal prediction.

Curriculum Learning Our approach is inspired from curriculum learning [101], where task difficulty gradually increases in complexity, propelling the policy to achieve incrementally harder subgoals. A genetic curriculum based approach [102] identifies unsolved scenarios to automatically generate an associated curriculum via adversarial training. ACCEL [103] proposes a regret based curriculum approach that keeps a record of previous scenarios, and selects the ones with highest regret. Prior work generates subgoals while considering the lower primitive performance, by factoring in the task success rate [104–106], value function [107, 108], achieved state density [123], and value uncertainty [124]. In contrast, our HRL based approach employs curriculum based learning for dealing with the issue of non-stationarity, by generating a curriculum of achievable subgoals.

5.4 Discussion

Limitations: Although adaptive relabeling might be an additional overhead in long-horizon tasks, we empirically found this to be a marginal overhead. Further, we plan to explore using undirected demonstrations in the future. Also, CRISP assumes the ability to reset the environment. A possible method for relaxing this assumption is to employ [125] that learns a backward controller to reset the environment.

Discussion: We introduce CRISP, a general purpose hierarchical curriculum based approach to deal with non-stationarity in HRL. CRISP parses expert demonstrations using our PIP approach to produce a curriculum of achievable subgoals for the lower primitive. We evaluate our method on complex sparsely rewarded robotic navigation and manipulation tasks in simulation, and show that our approach shows impressive performance and demonstrates substantial gains over baselines. We also show that CRISP demonstrates impressive performance in real world robotic tasks. We believe that hierarchical curriculum learning is a promising step towards building real world robotic systems.

Chapter 6

PEAR: Primitive Enabled Adaptive Relabeling for Boosting Hierarchical Reinforcement Learning

Example is not the main thing in influencing others. It is the only thing.

– Albert Schweitzer

¹ In the previous chapter, we proposed CRISP, which leverages expert demonstrations to generate efficient supervision for the hierarchical policies. It uses primitive informed parsing, where we select and relabel subgoals from expert demonstrations by allowing the lower level to execute for multiple time-steps, and selecting the failure states. This approach generates impressive results while assuming the ability to reset the environment to a previous state. However, this reset ability is generally not available for many tasks, especially when training in realistic scenarios, since apart from computational constraints, the noise in the physical world makes it almost impossible to exactly reset the environment

¹This chapter is based on the paper published previously at ICLR 2025 [78].

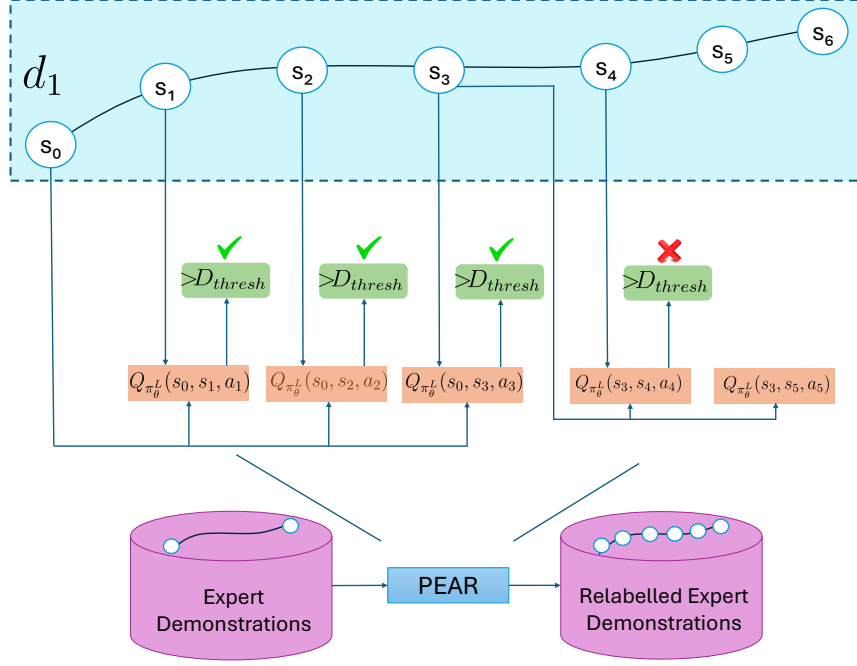


FIGURE 6.1: **Adaptive Relabeling Overview:** We segment expert demonstrations by consecutively passing demonstration states as subgoals to the lower primitive, and finding the state where $Q_{\pi^L}(s, s_i, a_i) < Q_{thresh}$ (here $s_i = s_4$). Since s_3 was the last reachable subgoal, it is selected as subgoal for initial state s_0 . The transition is added to D_g , and the process continues with s_3 as new initial state.

to a previous state. In this work, we leverage the lower level primitive’s Q function to adaptively relabel the expert demonstrations, thereby generating efficient supervision for the higher level policies.

The main motivation of this work is to produce a curriculum of feasible subgoals according to the current goal achieving capability of the lower primitive. Concretely, the value function of the lower primitive is used to perform *adaptive relabeling* on expert demonstrations to dynamically generate a curriculum of achievable subgoals for the lower primitive. This subgoal dataset is then used to train an imitation learning based regularizer, which is used to jointly optimize off-policy RL objective with IL regularization. Hence, our approach ameliorates non-stationarity in HRL by using primitive enabled IL regularization, while enabling efficient exploration using RL. We call our approach: *primitive enabled adaptive relabeling* (PEAR) for boosting HRL.

The major contributions of this work are: (i) our adaptive relabeling based approach

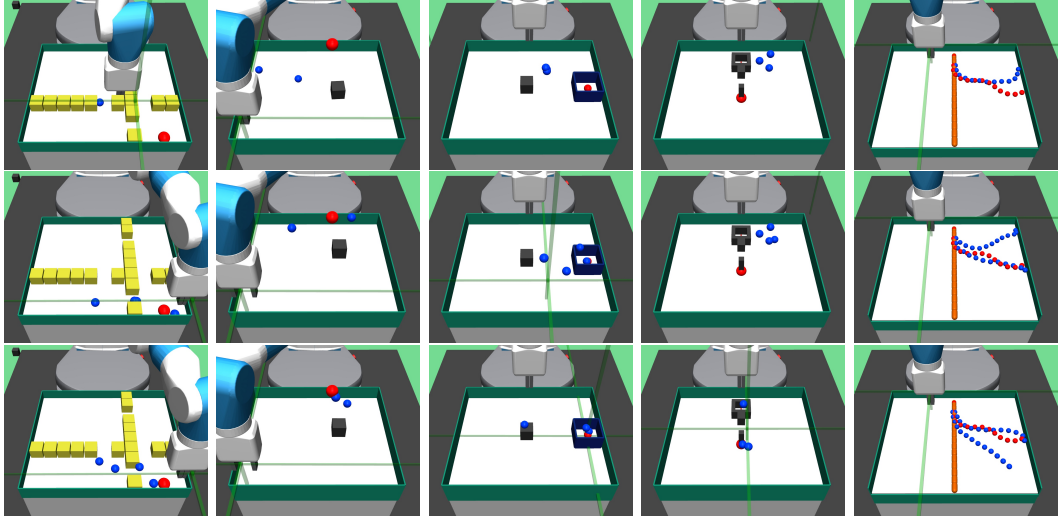


FIGURE 6.2: **Subgoal evolution:** With training, as the lower primitive improves, the higher level subgoal predictions (blue spheres) become better and harder, while always being achievable by lower primitive. Row 1 depicts initial training, Row 2 depicts mid-way through training, and Row 3 depicts end of training in maze navigation (Col 1), pick and place (Col 2), bin (Col 3), hollow (Col 4), rope manipulation (Col 5) and kitchen environment (Col 6). This generates a curriculum of achievable subgoals for lower primitive (red spheres: final goal).

generates efficient higher level subgoal supervision according to the current goal achieving capability of the lower primitive (Figure 6.2), (ii) we derive sub-optimality bounds to theoretically justify the benefits of periodic re-population using adaptive relabeling (Section 6.1.6), (iii) we perform extensive experimentation on sparse robotic tasks: maze navigation, pick and place, bin, hollow, rope manipulation and franka kitchen to empirically demonstrate superior performance and sample efficiency of PEAR over prior baselines (Section 6.2 Figure 6.3), and finally, (iv) we show that PEAR demonstrates impressive performance in real world tasks: pick and place, bin and rope manipulation (Section 6.2.

6.1 Primitive Enabled Adaptive Relabeling

We define our goal-conditioned off-policy RL setup as follows: *Universal Markov Decision Process* (UMDP) [72] is a Markov decision process augmented with the goal space G , where $M = (S, A, P, R, \gamma, G)$. Here, S is state space, A is action space, $P(s'|s, a)$ is the state transition probability function, R is reward function, and γ is discount factor. $\pi(a|s, g)$

represents the goal-conditioned policy which predicts the probability of taking action a when the state is s and goal is g . The overall objective is to maximize expected future discounted reward distribution: $J = (1 - \gamma)^{-1} \mathbb{E}_{s \sim d^\pi, a \sim \pi(a|s, g), g \sim G} [r(s_t, a_t, g)]$.

In our goal-conditioned **HRL** setup, the overall policy π is divided into multi-level policies. We consider bi-level scheme, where the higher level policy $\pi^H(s_g|s, g)$ predicts subgoals s_g for the lower primitive $\pi^L(a|s, s_g)$. π^H generates subgoals s_g after every c timesteps and π^L tries to achieve s_g within c timesteps. π^H gets sparse extrinsic reward r_{ex} from the environment, whereas π^L gets sparse intrinsic reward r_{in} from π^H . π^L gets rewarded with reward 0 if the agent reaches within δ^L distance of the predicted subgoal s_g , and -1 otherwise: $r_{in} = -1(\|s_t - s_g\|_2 > \delta^L)$. Similarly, π^H gets extrinsic reward 0 if the achieved goal is within δ^H distance of the final goal g , and -1 otherwise: $r_{ex} = -1(\|s_t - g\|_2 > \delta^H)$. We assume access to a small number of directed expert demonstrations $D = \{e^i\}_{i=1}^N$, where $e^i = (s_0^e, a_0^e, s_1^e, a_1^e \dots, s_{T-1}^e, a_{T-1}^e)$.

We explain **PEAR: Primitive Enabled Adaptive Relabeling** for boosting HRL, which leverages a few expert demonstrations D to solve long horizon tasks. We propose a two step approach: (i) the current lower primitive π^L is used to adaptively relabel expert demonstrations to generate efficient subgoal supervision D_g , and (ii) off-policy RL objective is jointly optimized with additional imitation learning based regularization objective using D_g . We also perform theoretical analysis to (i) bound the sub-optimality of our approach, and (ii) propose a practical generalized based framework for joint optimization using RL and IL, where typical off-policy RL and IL algorithms can be plugged in to generate various joint optimization based algorithms.

6.1.1 PEAR

PEAR performs adaptive relabeling on expert demonstration trajectories D to generate efficient higher level subgoal transition dataset D_g , by employing the *current* lower primitive's action value function $Q_{\pi^L}(s, s_i^e, a_i)$. In a typical goal-conditioned RL setting, $Q_{\pi^L}(s, s_i^e, a_i)$ describes the expected cumulative reward where the start state and subgoal

are s and s_i^e , and the lower primitive takes action a_i while following policy π^L . While parsing D , we consecutively pass the expert demonstration states s_i^e as subgoals, and $Q_{\pi^L}(s, s_i^e, a_i)$ computes the expected cumulative reward when the start state is s , subgoal is s_i^e and the next primitive action is a_i . Intuitively, a high value of $Q_{\pi^L}(s, s_i^e, a_i)$ implies that the current lower primitive considers s_i^e to be a good (highly rewarding and achievable) subgoal from current state s , since it expects to achieve a high intrinsic reward for this subgoal from the higher policy. Hence, $Q_{\pi^L}(s, s_i^e, a_i)$ considers goal achieving capability of current lower primitive for populating D_g . We depict a single pass of adaptive relabeling in Figure 6.1 and explain the procedure in detail below.

6.1.2 Adaptive Relabeling

Consider the demonstration dataset $D = \{e^j\}_{j=1}^N$, where each trajectory $e^j = (s_0^e, s_1^e, \dots, s_{T-1}^e)$. Let the initial state be s_0^e . In the adaptive relabeling procedure, we incrementally provide demonstration states s_i^e for $i = 1$ to $T - 1$ as subgoals to lower primitive’s action value function $Q_{\pi^L}(s_0^e, s_i^e, a_i)$, where $a_i = \pi^L(s = s_{i-1}^e, g = s_i^e)$. At every step, we compare $Q_{\pi^L}(s_0^e, s_i^e, a_i)$ to a threshold Q_{thresh} ($Q_{thresh} = 0$ works consistently for all experiments). If $Q_{\pi^L}(s_0^e, s_i^e, a_i) \geq Q_{thresh}$, we move on to next expert demonstration state s_{i+1}^e . Otherwise, we consider s_{i-1}^e to be a good subgoal (since it was the last subgoal with $Q_{\pi^L}(s_0^e, s_{i-1}^e, a_i) \geq Q_{thresh}$), and use it to compute subgoal transitions for populating D_g . Subsequently, we repeat the same procedure with s_{i-1}^e as the new initial state, until the episode terminates. This is depicted in Figure 6.1 and Algorithm 3.

6.1.3 Periodic re-population of higher level subgoal dataset

HRL approaches suffer from non-stationarity due to unstable higher level station transition and reward functions. In off-policy HRL, this occurs since previously collected experience is rendered obsolete due to continuously changing lower primitive. We propose to mitigate this non-stationarity by periodically re-populating subgoal transition dataset D_g after every p timesteps according to the goal achieving capability of the *current* lower primitive.

Since the lower primitive continuously improves with training and gets better at achieving harder subgoals, Q_{π^L} always picks reachable subgoals of appropriate difficulty, according to the current lower primitive. This generates a natural curriculum of achievable subgoals for the lower primitive. Intuitively, D_g always contains achievable subgoals for the current lower primitive, which mitigates non-stationarity in HRL. The pseudocode for PEAR is given in Algorithm 4. Figure 6.2 shows the qualitative evolution of subgoals with training in our experiments.

Algorithm 3 Adaptive Relabeling

```

1: Initialize  $D_g = \{\}$ 
2: // Populating  $D_g$ 
3: for each  $e = (s_0^e, s_1^e, \dots, s_{T-1}^e)$  in  $\mathcal{D}$  do
4:   Initial state index  $init \leftarrow 0$ 
5:   Subgoal transitions  $D_g^e = \{\}$ 
6:   for  $i = 1$  do  $T-1$ 
7:     // Find  $Q_{\pi^L}$  values for demo subgoals
8:     Compute  $Q_{\pi^L}(s_{init}^e, s_i^e, a_i)$ 
9:     where  $a_i = \pi^L(s_{i-1}^e, s_i^e)$ 
10:    // Find first subgoal s.t.  $Q_{\pi^L} < Q_{th}$ 
11:    if  $Q_{\pi^L}(s_{init}^e, s_i^e, a_i) < Q_{th}$  then
12:      for  $j = init$  to  $i - 1$  do
13:        for  $k = (init + 1)$  to  $i - 1$  do
14:          // Add the transition to  $D_g^e$ 
15:          Add  $(s_j, s_{i-1}, s_k)$  to  $D_g^e$ 
16:        Initial state index  $init \leftarrow (i - 1)$ 
17:    // Add selected transitions to  $D_g$ 
18:     $D_g \leftarrow D_g \cup D_g^e$ 

```

Algorithm 4 PEAR

```

1: Initialize  $D_g = \{\}$ 
2: for  $i = 1$  do  $N$ 
3:   if  $i \% p == 0$  then
4:     Clear  $D_g$ 
5:     Populate  $D_g$  via adaptive relabeling
6:   Collect experience using  $\pi^H$  and  $\pi^L$ 
7:   Update lower primitive via SAC and IL
8:   regularization with  $D_g^L$  (Eq 6.6 or Eq 6.8)
9:   Sample transitions from  $D_g$ 
10:  Update higher policy via SAC and IL
11:  regularization using  $D_g$  (Eq 6.5 or Eq 6.7)

```

6.1.4 Dealing with out-of-distribution states

Our adaptive relabeling procedure uses $Q_{\pi^L}(s_0^e, s_i^e, a_i)$ to select efficient subgoals when the expert state s_i^e is within the training distribution of states used to train the lower primitive. However, if the expert states are outside the training distribution, Q_{π^L} might erroneously over-estimate the values on such states, which might result in poor subgoal selection. In order to address this over-estimation issue, we employ an additional margin classification objective[126], where along with the standard Q_{SAC} objective, we also use an additional margin classification objective to yield the following optimization objective

$$\bar{Q}_{\pi^L} = Q_{SAC} +$$

$$\arg \min_{Q_{\pi^L}} \max_{\pi^L} (\mathbb{E}_{(s_0^e, \cdot, \cdot) \sim D_g, s_i^e \sim \pi^H, a_i \sim \pi^L} [Q_{\pi^L}(s_0^e, s_i^e, a_i)] - \mathbb{E}_{(s_0^e, s_i^e, \cdot) \sim D_g, a_i \sim \pi^L} [Q_{\pi^L}(s_0^e, s_i^e, a_i)])$$

This surrogate objective prevents over-estimation of $\bar{Q}_{\pi^L}$ by penalizing states that are out of the expert state distribution. We found this objective to improve performance and stabilize learning. In the next section, we explain how we use adaptive relabeling to yield our joint optimization objective.

6.1.5 Joint optimization

Here, we explain our joint optimization objective that comprises of off-policy RL objective with IL based regularization, using D_g generated using primitive enabled adaptive relabeling. We consider both behavior cloning (BC) and inverse reinforcement learning (IRL) regularization. Henceforth, PEAR-IRL will represent PEAR with IRL regularization and PEAR-BC will represent PEAR with BC regularization. We first explain BC regularization objective, and then explain IRL regularization objectives for both hierarchical levels.

For the BC objective, let $(s^e, s_g^e, s_{next}^e) \sim D_g$ represent a higher level subgoal transition from D_g where s^e is current state, s_{next}^e is next state, g^e is final goal and s_g^e is subgoal supervision. Let s_g be the subgoal predicted by the high level policy $\pi_{\theta^H}^H(\cdot | s^e, g^e)$ with

parameters θ_H . The BC regularization objective for higher level is as follows:

$$\min_{\theta_H} J_{BC}^H(\theta_H) = \min_{\theta_H} \mathbb{E}_{(s^e, s_g^e, s_{next}^e) \sim D_g, s_g \sim \pi_{\theta_H}^H(\cdot | s^e, g^e)} \|s_g^e - s_g\|^2 \quad (6.1)$$

Similarly, let $(s^f, a^f, s_{next}^f) \sim D_g^L$ represent lower level expert transition where s^f is current state, s_{next}^f is next state, g^f is goal and a is the primitive action predicted by $\pi_{\theta_L}^L(\cdot | s^f, s_g^e)$ with parameters θ_L . The lower level BC regularization objective is as follows:

$$\min_{\theta_L} J_{BC}^L(\theta_L) = \min_{\theta_L} \mathbb{E}_{(s^f, a^f, s_{next}^f) \sim D_g^L, a \sim \pi_{\theta_L}^L(\cdot | s^f, s_g^e)} \|a^f - a\|^2 \quad (6.2)$$

We now consider the IRL objective, which is implemented as a GAIL [127] objective implemented using LSGAN [128]. Let $\mathbb{D}_{\epsilon}^H$ be the higher level discriminator with parameters ϵ_H . Let J_D^H represent higher level IRL objective, which depends on parameters (θ_H, ϵ_H) . The higher level IRL regularization objective is as follows:

$$\begin{aligned} \max_{\theta_H} \min_{\epsilon_H} J_D^H(\theta_H, \epsilon_H) = \max_{\theta_H} \min_{\epsilon_H} & \frac{1}{2} \mathbb{E}_{(s^e, \cdot, \cdot) \sim D_g, s_g \sim \pi_{\theta_H}^H(\cdot | s^e, g^e)} [\mathbb{D}_{\epsilon_H}^H(\pi_{\theta_H}^H(\cdot | s^e, g^e)) - 0]^2 \\ & + \frac{1}{2} \mathbb{E}_{(s^e, s_g^e, \cdot) \sim D_g} [\mathbb{D}_{\epsilon_H}^H(s_g^e) - 1]^2 \end{aligned} \quad (6.3)$$

Similarly, for lower level primitive, let $\mathbb{D}_{\epsilon_L}^L$ be the lower level discriminator with parameters ϵ_L . Let J_D^L represent lower level IRL objective, which depends on parameters (θ_L, ϵ_L) . The lower level IRL regularization objective is as follows:

$$\begin{aligned} \max_{\theta_L} \min_{\epsilon_L} J_D^L(\theta_L, \epsilon_L) = \max_{\theta_L} \min_{\epsilon_L} & \frac{1}{2} \mathbb{E}_{(s^f, \cdot, \cdot) \sim D_g^L, a \sim \pi_{\theta_L}^L(\cdot | s^f, s_g^e)} [\mathbb{D}_{\epsilon_L}^L(\pi_{\theta_L}^L(\cdot | s^f, s_g^e)) - 0]^2 \\ & + \frac{1}{2} \mathbb{E}_{(s^f, a^f, \cdot) \sim D_g^L} [\mathbb{D}_{\epsilon_L}^L(a^f) - 1]^2 \end{aligned} \quad (6.4)$$

Finally, we describe our joint optimization objective for hierarchical policies. Let the off-policy RL objective be $J_{\theta_H}^H$ and $J_{\theta_L}^L$ for higher and lower policies. The joint optimization

objectives using BC regularization for higher and lower policies are provided in Equations 6.5 and 6.6 respectively.

$$\max_{\theta_H} (J_{\theta_H}^H - \psi * J_{BC}^H(\theta_H)) \quad (6.5)$$

$$\max_{\theta_L} (J_{\theta_L}^L - \psi * J_{BC}^L(\theta_L)) \quad (6.6)$$

The joint optimization objectives using IRL regularization for higher and lower policies are provided in Equations 6.7 and 6.8 respectively.

$$\min_{\epsilon_H} \max_{\theta_H} (J_{\theta_H}^H + \psi * J_D^H(\theta_H, \epsilon_H)) \quad (6.7)$$

$$\min_{\epsilon_L} \max_{\theta_L} (J_{\theta_L}^L + \psi * J_D^L(\theta_L, \epsilon_L)) \quad (6.8)$$

Here, ψ is regularization weight hyper-parameter. We describe ablations to choose ψ in Section 6.2.

6.1.6 Sub-optimality analysis

In this section, we perform theoretical analysis to (i) derive sub-optimality bounds for our proposed joint optimization objective and show how our periodic re-population based approach affects performance, and (ii) propose a generalized framework for joint optimization using RL and IL. Let π^* and π^{**} be unknown higher level and lower level optimal policies. Let $\pi_{\theta_H}^H$ be our high level policy and $\pi_{\theta_L}^L$ be our lower primitive policy, where θ_H and θ_L are trainable parameters. $D_{TV}(\pi_1, \pi_2)$ denotes total variation divergence between probability distributions π_1 and π_2 . Let κ be an unknown distribution over states and actions, G be goal space, s be current state, and g be the final episodic goal. We use κ in the importance sampling ratio later to avoid sampling from the unknown optimal policy (Appendix 6.1.7). The higher level policy predicts subgoals s_g for the lower primitive

which is executed for c timesteps to yield sub-trajectories τ . Let Π_D^H and Π_D^L be some unknown higher and lower level probability distributions over policies from which we can sample policies π_D^H and π_D^L . Let us assume that policies π_D^H and π_D^L represent the policies from higher and lower level datasets D_H and D_L respectively. Although D_H and D_L may represent any datasets, in our discussion, we use them to represent higher and lower level expert demonstration datasets. Firstly, we introduce the ϕ_D -common definition [129] in goal-conditioned policies:

Definition 1. π^* is ϕ_D -common in Π_D^H , if $\mathbb{E}_{\substack{\pi_D^H \sim \Pi_D^H \\ s \sim \kappa \\ g \sim G}}[D_{TV}(\pi^*(\tau|s, g) || \pi_D^H(\tau|s, g))] \leq \phi_D$

Now, we define the suboptimality of policy π with respect to optimal policy π^* as:

$$Subopt(\theta) = |J(\pi^*) - J(\pi)| \quad (6.9)$$

Theorem 1. Assuming optimal policy π^* is ϕ_D common in Π_D^H , the suboptimality of higher policy $\pi_{\theta_H}^H$, over c length sub-trajectories τ sampled from $d_c^{\pi^*}$ can be bounded as:

$$|J(\pi^*) - J(\pi_{\theta_H}^H)| \leq \underbrace{\lambda_H * \phi_D}_{\text{first term}} + \underbrace{\lambda_H * \mathbb{E}_{s \sim \kappa, \pi_D^H \sim \Pi_D^H, g \sim G}[D_{TV}(\pi_D^H(\tau|s, g) || \pi_{\theta_H}^H(\tau|s, g))]}_{\text{second term}} \quad (6.10)$$

where $\lambda_H = \frac{2}{(1-\gamma)(1-\gamma^c)} R_{max} \left\| \frac{d_c^{\pi^*}}{\kappa} \right\|_{\infty}$

Similarly, the suboptimality of lower primitive $\pi_{\theta_L}^L$ can be bounded as:

$$|J(\pi^{**}) - J(\pi_{\theta_L}^L)| \leq \lambda_L * \phi_D + \lambda_L * \mathbb{E}_{s \sim \kappa, \pi_D^L \sim \Pi_D^L, s_g \sim \pi_{\theta_H}^H}[D_{TV}(\pi_D^L(\tau|s, s_g) || \pi_{\theta_L}^L(\tau|s, s_g))] \quad (6.11)$$

where $\lambda_L = \frac{2}{(1-\gamma)^2} R_{max} \left\| \frac{d_c^{\pi^{**}}}{\kappa} \right\|_{\infty}$

The proofs for Equations 6.10 and 6.11 are provided as follows:

6.1.7 Proofs for Theorem 1 for higher and lower level policies

Here, we present the proofs for Theorem 1 for higher and lower level policies, which provide sub-optimality bounds on the optimization objectives.

6.1.7.1 Sub-optimality proof for higher level policy

The sub-optimality of upper policy $\pi_{\theta_H}^H$, over c length sub-trajectories τ sampled from $d_c^{\pi^*}$ can be bounded as:

$$|J(\pi^*) - J(\pi_{\theta_H}^H)| \leq \lambda_H * \phi_D + \lambda_H * \mathbb{E}_{s \sim \kappa, \pi_D^H \sim \Pi_D^H, g \sim G} [D_{TV}(\pi_D^H(\tau|s, g) || \pi_{\theta_H}^H(\tau|s, g))] \quad (6.12)$$

where $\lambda_H = \frac{2}{(1-\gamma)(1-\gamma^c)} R_{max} \left\| \frac{d_c^{\pi^*}}{\kappa} \right\|_{\infty}$

Proof. We extend the suboptimality bound from [129] between goal conditioned policies π^* and $\pi_{\theta_H}^H$ as follows:

$$|J(\pi^*) - J(\pi_{\theta_H}^H)| \leq \frac{2}{(1-\gamma)(1-\gamma^c)} R_{max} \mathbb{E}_{s \sim d_c^{\pi^*}, g \sim G} [D_{TV}(\pi^*(\tau|s, g) || \pi_{\theta_H}^H(\tau|s, g))] \quad (6.13)$$

By applying triangle inequality:

$$D_{TV}(\pi^*(\tau|s, g) || \pi_{\theta_H}^H(\tau|s, g)) \leq D_{TV}(\pi^*(\tau|s, g) || \pi_D^H(\tau|s, g)) + D_{TV}(\pi_D^H(\tau|s, g) || \pi_{\theta_H}^H(\tau|s, g)) \quad (6.14)$$

Taking expectation wrt $s \sim \kappa$, $g \sim G$ and $\pi_D^H \sim \Pi_D^H$,

$$\begin{aligned} \mathbb{E}_{s \sim \kappa, g \sim G} [D_{TV}(\pi^*(\tau|s, g) || \pi_{\theta_H}^H(\tau|s, g))] &\leq \mathbb{E}_{s \sim \kappa, \pi_D^H \sim \Pi_D^H, g \sim G} [D_{TV}(\pi^*(\tau|s, g) || \pi_D^H(\tau|s, g))] + \\ &\quad \mathbb{E}_{s \sim \kappa, \pi_D^H \sim \Pi_D^H, g \sim G} [D_{TV}(\pi_D^H(\tau|s, g) || \pi_{\theta_H}^H(\tau|s, g))] \end{aligned} \quad (6.15)$$

Since π^* is ϕ_D common in Π_D^H , we can write 6.15 as:

$$\begin{aligned} \mathbb{E}_{s \sim \kappa, g \sim G}[D_{TV}(\pi^*(\tau|s, g)||\pi_{\theta_H}^H(\tau|s, g))] &\leq \\ \phi_D + \mathbb{E}_{s \sim \kappa, \pi_D^H \sim \Pi_D^H, g \sim G}[D_{TV}(\pi_D^H(\tau|s, g)||\pi_{\theta_H}^H(\tau|s, g))] \end{aligned} \quad (6.16)$$

Substituting the result from align 6.16 in align 6.13, we get

$$|J(\pi^*) - J(\pi_{\theta_H}^H)| \leq \lambda_H * \phi_D + \lambda_H * \mathbb{E}_{s \sim \kappa, \pi_D^H \sim \Pi_D^H, g \sim G}[D_{TV}(\pi_D^H(\tau|s, g)||\pi_{\theta_H}^H(\tau|s, g))] \quad (6.17)$$

where $\lambda_H = \frac{2}{(1-\gamma)(1-\gamma^c)} R_{max} \|\frac{d_c^{\pi^*}}{\kappa}\|_\infty$ □

6.1.7.2 Sub-optimality proof for lower level policy

Let the optimal lower level policy be π^{**} . The suboptimality of lower primitive $\pi_{\theta_L}^L$ can be bounded as follows:

$$|J(\pi^{**}) - J(\pi_{\theta_L}^L)| \leq \lambda_L * \phi_D + \lambda_L * \mathbb{E}_{s \sim \kappa, \pi_D^L \sim \Pi_D^L, s_g \sim \pi_{\theta_H}^H}[D_{TV}(\pi_D^L(\tau|s, s_g)||\pi_{\theta_L}^L(\tau|s, s_g))] \quad (6.18)$$

where $\lambda_L = \frac{2}{(1-\gamma)^2} R_{max} \|\frac{d_c^{\pi^{**}}}{\kappa}\|_\infty$

Proof. We extend the suboptimality bound from [129] between goal conditioned policies π^{**} and $\pi_{\theta_L}^L$ as follows:

$$|J(\pi^{**}) - J(\pi_{\theta_L}^L)| \leq \frac{2}{(1-\gamma)^2} R_{max} \mathbb{E}_{s \sim d_c^{\pi^{**}}, s_g \sim \pi_{\theta_H}^H}[D_{TV}(\pi^{**}(\tau|s, s_g)||\pi_{\theta_L}^L(\tau|s, s_g))] \quad (6.19)$$

By applying triangle inequality:

$$\begin{aligned} D_{TV}(\pi^{**}(\tau|s, s_g)||\pi_{\theta_L}^L(\tau|s, s_g)) &\leq D_{TV}(\pi^{**}(\tau|s, s_g)||\pi_D^L(\tau|s, s_g)) + \\ &\quad D_{TV}(\pi_D^L(\tau|s, s_g)||\pi_{\theta_L}^L(\tau|s, s_g)) \end{aligned} \quad (6.20)$$

Taking expectation wrt $s \sim \kappa$, $s_g \sim \pi_{\theta_H}^H$ and $\pi_D^L \sim \Pi_D^L$,

$$\begin{aligned} & \mathbb{E}_{s \sim \kappa, s_g \sim \pi_{\theta_H}^H} [D_{TV}(\pi^{**}(\tau|s, s_g) || \pi_{\theta_L}^L(\tau|s, s_g))] \leq \\ & \mathbb{E}_{s \sim \kappa, \pi_D^L \sim \Pi_D^L, s_g \sim \pi_{\theta_H}^H} [D_{TV}(\pi^{**}(\tau|s, s_g) || \pi_D^L(\tau|s, s_g))] + \\ & \mathbb{E}_{s \sim \kappa, \pi_D^L \sim \Pi_D^L, s_g \sim \pi_{\theta_H}^H} [D_{TV}(\pi_D^L(\tau|s, s_g) || \pi_{\theta_L}^L(\tau|s, s_g))] \end{aligned} \quad (6.21)$$

Since π^{**} is ϕ_D common in Π_D^L , we can write 6.21 as:

$$\begin{aligned} & \mathbb{E}_{s \sim \kappa, s_g \sim \pi_{\theta_H}^H} [D_{TV}(\pi^{**}(\tau|s, s_g) || \pi_{\theta_L}^L(\tau|s, s_g))] \leq \\ & \phi_D + \mathbb{E}_{s \sim \kappa, \pi_D^L \sim \Pi_D^L, s_g \sim \pi_{\theta_H}^H} [D_{TV}(\pi_D^L(\tau|s, s_g) || \pi_{\theta_L}^L(\tau|s, s_g))] \end{aligned} \quad (6.22)$$

Substituting the result from align 6.22 in align 6.19, we get

$$|J(\pi^{**}) - J(\pi_{\theta_L}^L)| \leq \lambda_L * \phi_D + \lambda_L * \mathbb{E}_{s \sim \kappa, \pi_D^L \sim \Pi_D^L, s_g \sim \pi_{\theta_H}^H} [D_{TV}(\pi_D^L(\tau|s, s_g) || \pi_{\theta_L}^L(\tau|s, s_g))] \quad (6.23)$$

where $\lambda_L = \frac{2}{(1-\gamma)^2} R_{max} \| \frac{d_c^{\pi^{**}}}{\kappa} \|_{\infty}$ □

We next discuss the effect of training on the two terms in RHS of Equation 6.10, which bound the suboptimality of $\pi_{\theta_H}^H$.

Effect of adaptive relabeling on sub-optimality bounds

We firstly focus on the first term which is dependent on ϕ_D . Since we represent the generated subgoal dataset as D_g , we replace ϕ_D with ϕ_{D_g} . In Theorem 1, we assume the optimal policy π^* to be ϕ_{D_g} common in Π_D^H . Since ϕ_{D_g} denotes the upper bound of the expected TV divergence between π^* and π_D^H , ϕ_{D_g} provides a quality measure of the subgoal dataset D_g populated using adaptive relabeling. Intuitively, a lower value of ϕ_{D_g} implies that the optimal policy π^* is closely represented by D_g , or alternatively, the samples from D_g are near optimal. As the lower primitive improves with training and is able to achieve harder subgoals, and since D_g is re-populated using the improved lower primitive after every p timesteps, π_{D_g} continually gets closer to π^* , which results in reduced value of ϕ_D . This

implies that due to decreasing first term, the suboptimality bound in Equation 6.10 gets tighter, and consequently $J(\pi_{\theta_H}^H)$ gets closer to optimal $J(\pi^*)$ objective. Hence, our periodic re-population based approach generates a natural curriculum of achievable subgoals for the lower primitive, which continuously improves the performance by tightening the upper bound.

Effect of IL regularization on sub-optimality bounds

Now, we focus on the second term in Equation 6.10, which is TV divergence between $\pi_D^H(\tau|s, g)$ and $\pi_{\theta_H}^H(\tau|s, g)$ with expectation over $s \sim \kappa, \pi_D^H \sim \Pi_D^H, g \sim G$. As before, D is replaced by dataset D_g . This term can be viewed as imitation learning (IL) objective between expert demonstration policy $\pi_{D_g}^H$ and current policy $\pi_{\theta_H}^H$, where TV divergence is the distance measure. Due to this IL regularization objective, as policy $\pi_{\theta_H}^H$ gets closer to expert distribution policy $\pi_{D_g}^H$ with training, the LHS sub-optimality bounds get tighter. Thus, our proposed periodic IL regularization using D_g tightens the sub-optimality bounds in Equation 6.10 with training, thereby improving performance.

Generalized framework

We now derive our generalized framework for the joint optimization objective, where we can plug in off-the-shelf RL and IL methods to yield a generally applicable practical HRL algorithm. Considering sub-optimality is positive (Equation 6.9), we can use Equation 6.10 to derive the following objective:

$$J(\pi^*) \geq \underbrace{J(\pi_{\theta_H}^H)}_{\text{RL term}} - \underbrace{\lambda_H * \phi_D}_{\text{const. wrt } D_g} - \underbrace{\lambda_H * \mathbb{E}_{s \sim \kappa, \pi_D^H \sim \Pi_D^H, g \sim G} [d(\pi_D^H(\tau|s, g) || \pi_{\theta_H}^H(\tau|s, g))]}_{\text{IL regularization term}} \quad (6.24)$$

where (considering $\pi_D^H(\tau|s, g)$ as π_A , and $\pi_{\theta_H}^H(\tau|s, g)$ as π_B), $d(\pi_A || \pi_B) = D_{TV}(\pi_A || \pi_B)$.

Notably, the second term $\lambda_H * \phi_D$ in RHS of Equation 6.24 is constant for a given dataset D_g . Equation 6.24 can be perceived as a minorize maximize algorithm which intuitively means: the overall objective can be optimized by (i) maximizing the objective $J(\pi_{\theta_H}^H)$ via RL, and (ii) minimizing the distance measure d between π_D^H and $\pi_{\theta_H}^H$ (IL regularization). This formulation serves as a framework where we can plug in RL algorithm of choice for

off-policy RL objective $J(\pi_{\theta_H}^H)$, and distance function d of choice for IL regularization, to yield various joint optimization objectives.

In our setup, we plug in entropy regularized Soft Actor Critic [130] to maximize $J(\pi_{\theta_H}^H)$. Notably, different parameterizations of d yield different imitation learning regularizers. When d is formulated as Kullback–Leibler divergence, the IL regularizer takes the form of behavior cloning (BC) objective [131] (which results in PEAR-BC), and when d is formulated as Jensen-Shannon divergence, the imitation learning objective takes the form of inverse reinforcement learning (IRL) objective (which results in PEAR-IRL). We consider both these objectives in Section 6.2 and provide empirical performance results.

6.2 Experiments

In this section, we empirically answer the following questions: *(i)* does adaptive relabeling approach outperform fixed relabeling based approaches, *(ii)* is PEAR able to mitigate non-stationarity, *(iii)* does IL regularization boost performance in solving complex long horizon tasks, and *(iv)* What is the contribution of each of our design choices? We accordingly perform experiments on six Mujoco [88] environments: *(i)* maze navigation, *(ii)* pick and place, *(iii)* bin, *(iv)* hollow, *(v)* rope manipulation, and *(vi)* franka kitchen.

Environment and Implementation Details: Here, we provide extensive environment and implementation details for various environments. We perform the experiments on two system each with Intel Core i7 processors, equipped with 48GB RAM and Nvidia GeForce GTX 1080 GPUs. We use 28 expert demos for franks kitchen task and 100 demos in all other tasks. We empirically increased the number of demonstrations until there was no significant improvement in the performance. In our experiments, we use Soft Actor Critic [132]. The actor, critic and discriminator networks are formulated as 3 layer fully connected networks with 512 neurons in each layer.

When calculating p , we normalize Q_{π^L} values of a trajectory before comparing with Q_{thresh} : $((Q_{\pi^L}(s_0^e, s_i^e, a_i) - min_value)/max_value) * 100$ for $i = 1$ to $T - 1$. The experiments are

run for $4.73e5$, $1.1e5$, $1.32E5$, $1.8E5$, $1.58e6$, and $5.32e5$ timesteps in maze, pick and place, bin, hollow, rope and kitchen respectively. The regularization weight hyper-parameter Ψ is set at 0.001, 0.005, 0.005, 0.005, 0.005, and 0.005, the population hyper-parameter p is set to be $1.1e4$, 2500, 2500, 2500, $3.9e5$, and $1.4e4$, and distance threshold hyper-parameter Q_{thresh} is set at 10, 0, 0, 0, 0, and 0 for maze, pick and place, bin, hollow, rope and kitchen tasks respectively. Since the environments are sparsely rewarded, they are complex tasks where the agent must explore the environment extensively before receiving any rewards. Unless otherwise stated, we keep the training conditions consistent across all baselines to ascertain fair comparisons, and empirically tune the hyper-parameter values of our method and all other baselines.

6.2.1 Evaluation and Results

In Figure 6.3, we depict the success rate performance of PEAR and compare it with other baselines averaged over 5 seeds. The primary goal of these comparisons is to verify that the proposed approach indeed mitigates non-stationarity and demonstrates improved performance and training stability.

Comparing with fixed window based approach

RPL: In order to demonstrate the efficacy of adaptive relabeling, we compare our method with Relay Policy Learning (RPL) baseline. RPL [89] uses supervised pre-training followed by relay fine tuning. In order to ascertain fair comparisons, we use an ablation of RPL which does not use supervised pre-training. PEAR outperforms this baseline, which demonstrates that adaptive relabeling outperforms fixed window based relabeling and is crucial for mitigating non-stationarity. Although both approaches utilize reinforcement learning with imitation learning regularization, the primary distinction lies in how they collect subgoals. PEAR employs an adaptive relabeling mechanism that dynamically identifies subgoals within the reach of the current lower-level primitive, enabling the system to adjust subgoal complexity automatically without requiring manual intervention. Thus, it is evident that adaptive relabeling is crucial for generating feasible subgoals.

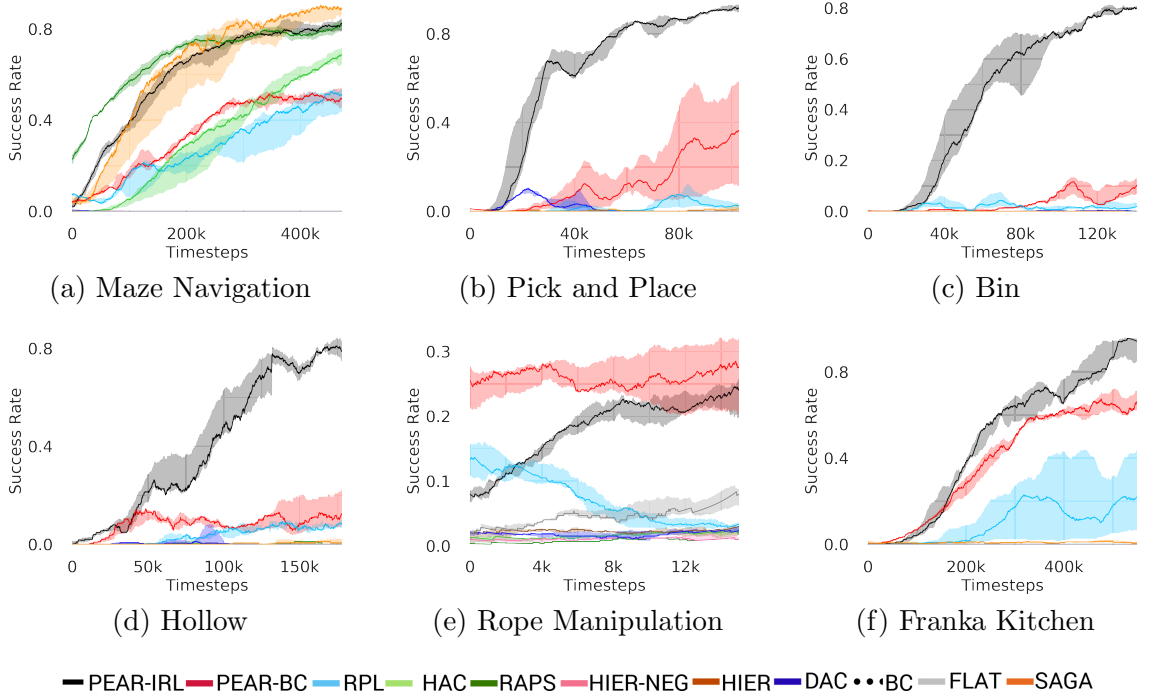


FIGURE 6.3: **Success rate comparison** This figure compares the success rate performances on six sparse robotic environments: Maze Navigation (a), Pick and Place (b), Bin (c), Hollow (d), Rope Manipulation (e), and Franka Kitchen environment (f). The solid line and shaded region represent the mean and range of success rates across 5 seeds. As seen, PEAR shows impressive performance and significantly outperforms the baselines.

Comparing with hierarchical baselines

SAGA baseline: We compare PEAR with SAGA, which is a hierarchical approach that employs state conditioned discriminator network training to address non-stationarity, by ensuring that the high-level policy generates subgoals that align with the current state of the low-level policy. We find that although SAGA performs well in maze navigation environment, it fails to solve harder tasks, whereas PEAR is able to significantly outperform the baseline. This demonstrates that SAGA suffers from non-stationarity issue in harder long horizon tasks, and PEAR is able to better mitigate non-stationarity issue and effectively solve long horizon tasks.

RAPS: RAPS [133] uses hand designed action primitives at the lower level, where the goal of the upper level is to pick the optimal sequence of action primitives. The performance of such approaches significantly depends on the quality of action primitives, which require substantial effort to hand-design. We found that except maze navigation task, RAPS is

unable to perform well on other tasks, which we believe is because selecting appropriate primitive sequences is hard on other harder tasks. Notably, hand designing action primitives is exceptionally complex in environments like rope manipulation. Hence, we do not evaluate RAPS in rope environment. In contrast, PEAR outperforms RAPS on harder environments, demonstrating the advantage of hierarchical abstraction and adaptive relabeling based subgoal regularization. Notably, PEAR is much more general-purpose since unlike RAPS, PEAR does not require domain-specific action primitives.

HAC: Hierarchical actor critic (HAC) [134] deals with non-stationarity by relabeling transitions while assuming an optimal lower primitive. Although HAC shows good performance in maze navigation task, PEAR consistently outperforms HAC on all other tasks.

As shown in Figure 6.3, while HAC attains competitive results on relatively simple maze navigation tasks, it faces difficulties in more complex environments. In contrast, PEAR employs an adaptive subgoal relabeling strategy that dynamically calibrates subgoal difficulty in accordance with the evolving ability of the lower-level policy. This approach facilitates more stable hierarchical training, ensures the generation of achievable subgoals, and consequently delivers improved sample efficiency and higher success rates across diverse tasks.

HIER-NEG and HIER: We also compare PEAR with two hierarchical baselines: HIER and HIER-NEG, which are hierarchical baselines that do not leverage expert demonstrations. HIER-NEG is a hierarchical baseline where the upper level is negatively rewarded if the lower primitive fails to achieve the subgoal. Since HIER, HIER-NEG and PEAR all are hierarchical approaches, we use these baseline comparisons to motivate that the performance improvement is not just due to use of hierarchical abstraction, but instead due to adaptive relabeling and primitive-enabled regularization. This is clearly evidenced by superior performance of PEAR.

Comparing with non-hierarchical baselines

Additionally, we consider single-level Discriminator Actor Critic (DAC) [135] that leverages expert demos, single-level SAC (FLAT) baseline, and Behavior Cloning (BC) baselines.

These baselines consistently fail to reliably solve the evaluated tasks, underscoring the essential role of combining hierarchical abstraction with curriculum-driven subgoal adaptation.

6.2.2 Qualitative Insights

While quantitative comparisons (e.g., success rates, learning curves) demonstrate the clear empirical superiority of PEAR, we now provide qualitative insights to elucidate **how and why** PEAR’s hierarchical formulation and adaptive subgoal curriculum confer concrete advantages over prior deep RL baselines.

- Adaptive Subgoal Curriculum:** Unlike baselines that use fixed window relabeling or hand-designed subgoal primitives, PEAR dynamically selects subgoals based on the *current* ability of the low-level (primitive) policy. As illustrated in Fig. 6.2, early in training, the high-level policy produces simple, incremental subgoals that are always within reach for the low-level policy. As training progresses, these subgoals naturally become more challenging, yielding a curriculum effect. This adaptive mechanism ensures continuous progress and avoids the degenerate behaviors (e.g., stalling, cycling over unreachable subgoals) observed in fixed-segmentation methods.
- Interpretability and Task Decomposition:** We observe that the hierarchical structure enables the agent to break down long-horizon tasks into interpretable intermediate steps. For instance, in *pick-and-place* and *bin* tasks, the high-level policy first guides the agent to approach an object, then instructs gripping, and finally targets transport to the goal destination. Such explicit stepwise decomposition is rarely observed in flat RL or non-adaptive hierarchies, which often attempt these tasks in a monolithic and less sample-efficient manner.

- Mitigating Non-Stationarity and Infeasible Subgoals:** The periodic re-population of subgoal datasets based on the low-level value function ensures that high-level instructions remain feasible as the low-level policy evolves. This is reflected in consistently low distances between proposed and achieved subgoals throughout training (Fig. 6.4), indicating that subgoals are within the agent’s actual capability, thereby enhancing training stability and exploration efficiency. In contrast, alternative hierarchical and non-hierarchical baselines frequently issued unattainable subgoals, leading to non-stationarity and stagnation.
- Failure Modes of Baselines:** During evaluation, we observed that fixed segmentation methods (e.g., RPL) sometimes propose subgoals that are either trivial (no significant task progress) or infeasible (beyond the reach of the low-level), resulting in poor downstream performance. In contrast, PEAR’s adaptive regularization systematically prevents such failure modes by grounding each subgoal in the demonstrated capabilities of the evolving primitive policy.

These results and observations collectively underscore that PEAR not only offers quantitative gains but also fundamentally alters the *quality* of agent behavior by exploiting the synergy of hierarchy and adaptive subgoal setting. We hope these insights clarify why our approach outperforms both traditional and recent deep RL alternatives, providing a foundation for future hierarchical RL research.

6.2.3 Ablative analysis

Here, we perform ablation analysis to elucidate the significance of our design choices. We choose the hyper-parameter values after extensive experiments, and keep them consistent across all baselines.

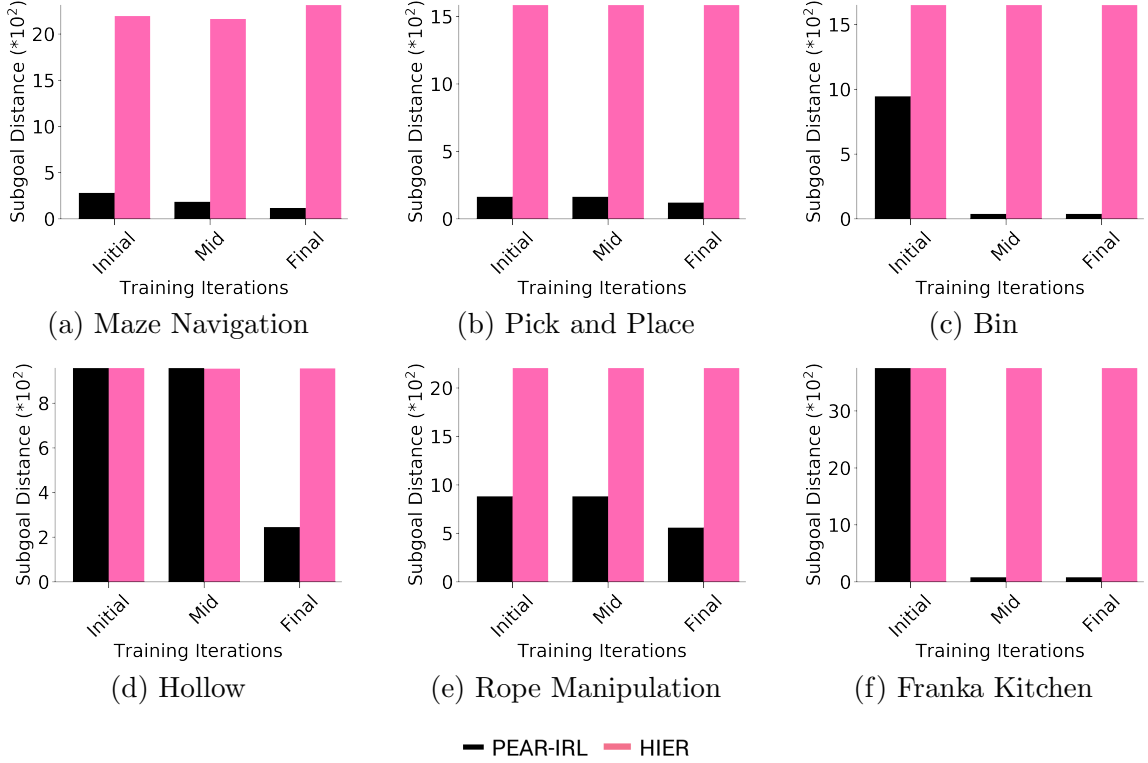


FIGURE 6.4: **Non-stationarity metric comparison** This figure compares the average distance metric between the subgoals predicted by the higher level policy and the subgoals achieved by the lower level policy during training on six sparse robotic environments: Maze Navigation (a), Pick and Place (b), Bin (c), Hollow (d), Rope Manipulation (e), and Franka Kitchen environment (f). As seen, PEAR consistently produces efficient subgoals leading to low distances between the predicted and achieved subgoals throughout the training process. This mitigates non-stationarity in HRL.

6.2.3.1 Dealing with non-stationarity and infeasible subgoal generation in HRL

We assess whether PEAR mitigates non-stationarity in HRL by comparing it with the vanilla HIER baseline, as shown in Figure 6.4. We compute the average distance between subgoals predicted by the higher-level policy and those achieved by the lower-level primitive throughout training. A lower average distance suggests that PEAR generates subgoals achievable by the lower primitive, inducing lower primitive behavior to be optimal. Our findings reveal that PEAR consistently maintains low average distances, validating its effectiveness in reducing non-stationarity. Additionally, as seen in Figure 6.5, post-training results show that PEAR achieves significantly lower distance values than the HIER baseline, highlighting its ability to generate feasible subgoals through primitive regularization.

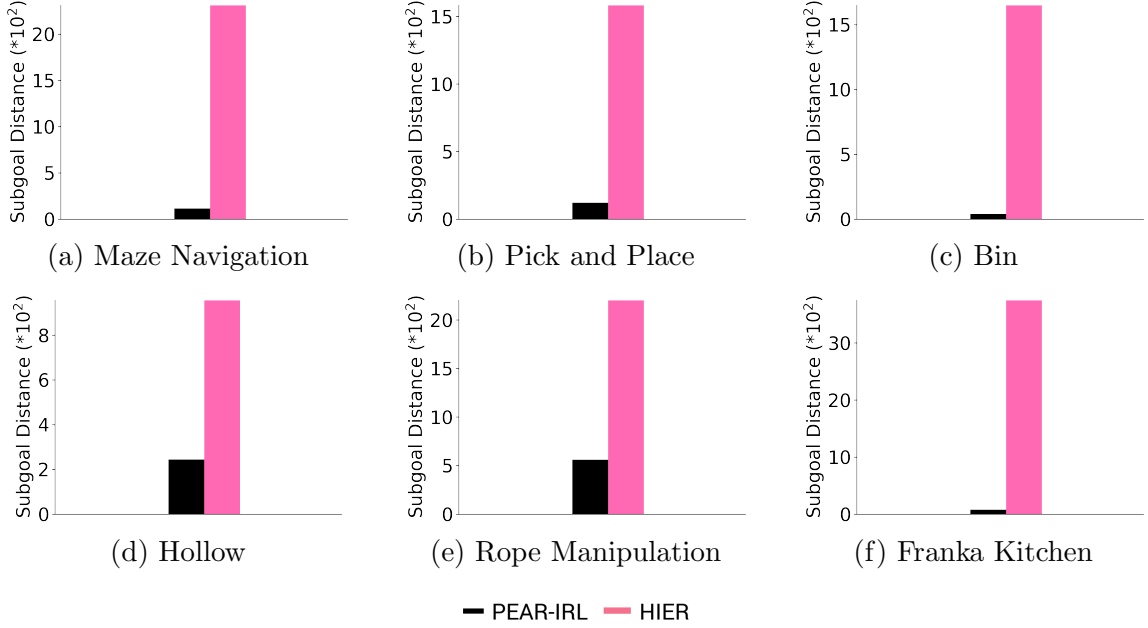


FIGURE 6.5: **Feasible subgoal generation metric comparison** This figure compares the average distance metric between the subgoals predicted by the higher level policy and the subgoals achieved by the lower level policy after the training is completed, on six sparse robotic environments: Maze Navigation (a), Pick and Place (b), Bin (c), Hollow (d), Rope Manipulation (e), and Franka Kitchen environment (f). As seen in the figure, PEAR consistently produces feasible subgoals which are achievable by the lower level primitive.

6.2.3.2 Additional Ablations

First, we verify the importance of adaptive relabeling by replacing it in PEAR-IRL by fixed window relabeling (as in RPL [89]). As seen in Figure 6.6, this ablation (PEAR-IRL) consistently outperforms PEAR-RPL on all tasks, which demonstrates the benefit of adaptive relabeling. Further, we compare PEAR-IRL and PEAR-BC (with margin classification objectives), with PEAR-IRL-No-Margin and PEAR-BC-No-Margin (without margin objectives) in Figure 6.7. PEAR-IRL and PEAR-BC outperform their No-Margin counterparts, which shows that this objective efficiently deals with the issue of out-of-distribution states, and induces training stability.

Further, we analyse how varying Q_{thresh} affects performance in Figure 6.8. We next study the impact of varying p . Intuitively, if p is too large, it impedes generation of a good curriculum of subgoals (Figure 6.9). Also, a low value of p may lead to frequent subgoal dataset re-population and may impede stable learning. We also choose optimal window

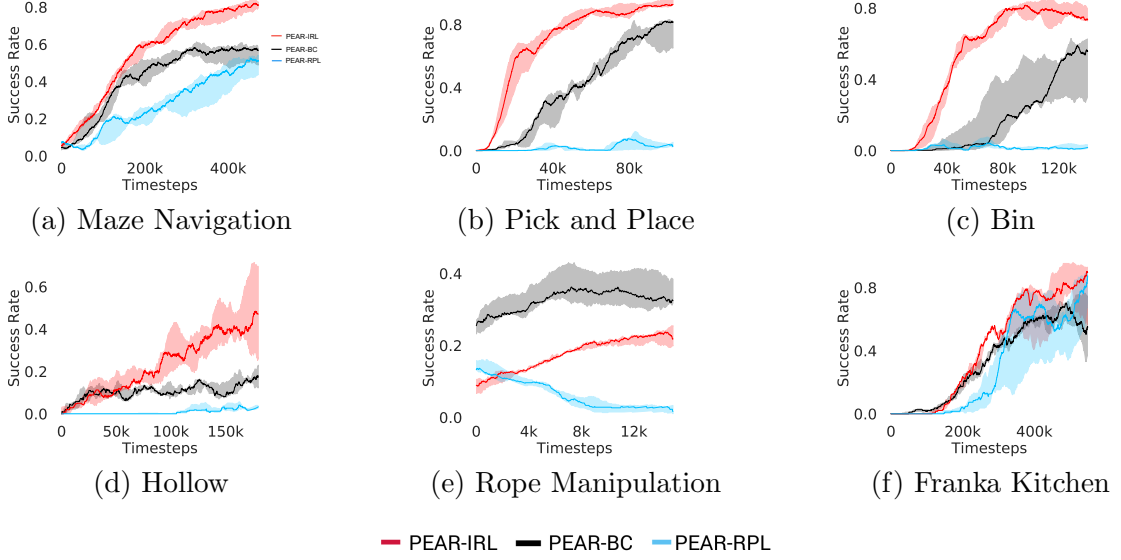


FIGURE 6.6: The success rate plots show success rate performance comparison between PEAR-IRL (red), PEAR-BC (black) and PEAR-RPL (blue) ablation on six sparse robotic environments: Maze Navigation (a), Pick and Place (b), Bin (c), Hollow (d), Rope Manipulation (e), and Franka Kitchen environment (f). As seen in the figure, PEAR-IRL and PEAR-BC clearly outperform PEAR-RPL in almost all the tasks.

size k for RPL experiments, as shown in Figure 6.10. We also evaluate learning rate ψ in Figure 6.11. If ψ is too small, PEAR is unable to utilize IL regularization, whereas conversely if ψ is too large, the learned policy might overfit. We also deduce the optimal number of expert demonstrations required in Figure 6.12. Next, we compare the performance of PEAR-IRL with HER-BC, which is a single-level implementation of HER with expert demonstrations. As seen in Figure 6.13, PEAR significantly outperforms this baseline, which demonstrates the advantage of our hierarchical formulation.

6.2.3.3 Real world experiments

We perform experiments on real world robotic pick and place, bin and rope environments. We use Realsense D435 depth camera to extract the robotic arm position, block, bin, and rope cylinder positions. Computing accurate linear and angular velocities is hard in real tasks, so we assign them small hard-coded values, which shows good performance. We performed 5 sets of experiments with 10 trial each, and report the average success rates. PEAR-IRL achieves accuracy of 0.8, 0.6, and 0.3, whereas PEAR-BC achieves accuracy of 0.8,

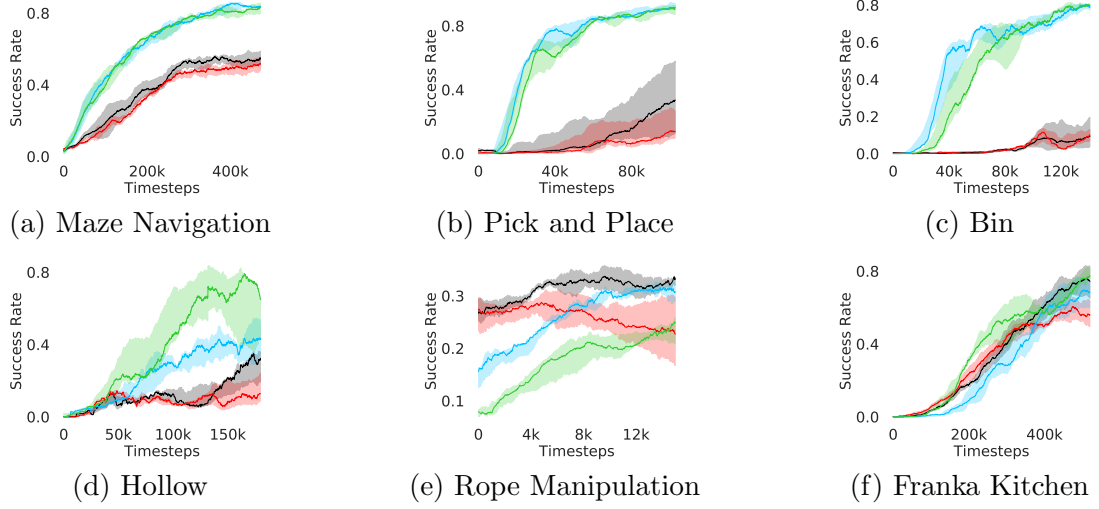


FIGURE 6.7: This figure illustrates the importance of the margin surrogate objective by comparing PEAR-IRL and PEAR-BC (with margin objective) with PEAR-IRL-No-Margin and PEAR-BC-No-Margin (without margin objective) on six sparse robotic environments: Maze Navigation (a), Pick and Place (b), Bin (c), Hollow (d), Rope Manipulation (e), and Franka Kitchen environment (f). As seen in the figure, PEAR-IRL and PEAR-BC outperform their no-margin counterparts in almost all tasks.

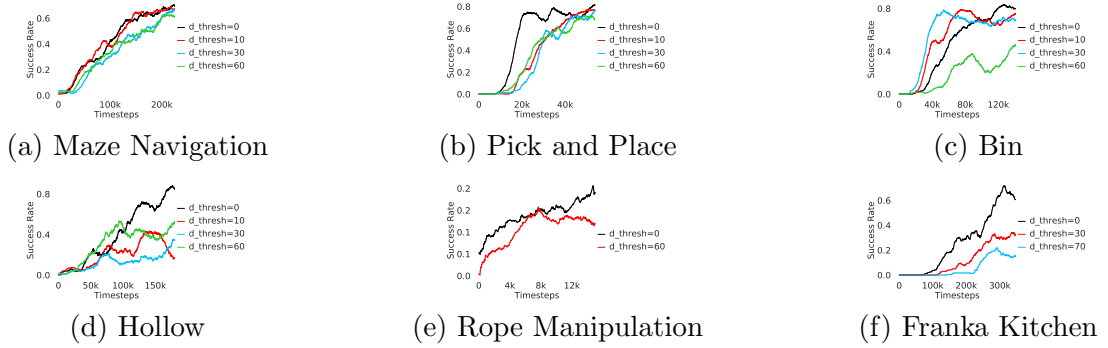


FIGURE 6.8: This figure shows the performance of PEAR for various values of the Q_{thresh} parameter versus number of training timesteps on six sparse robotic environments: Maze Navigation (a), Pick and Place (b), Bin (c), Hollow (d), Rope Manipulation (e), and Franka Kitchen environment (f).

0, 0.3 on pick and place, bin and rope environments. We also evaluate the performance of next best performing RPL baseline, but it fails to achieve success in any of the tasks.

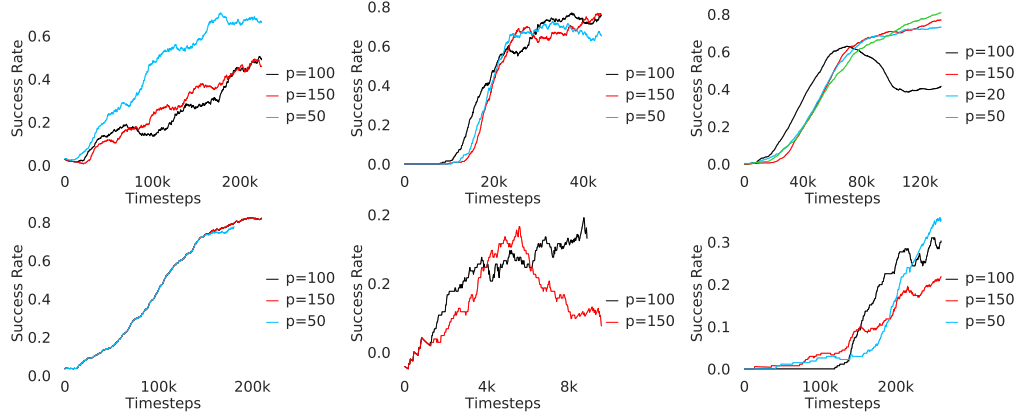


FIGURE 6.9: The success rate plots show the performance of PEAR for various values of population number p parameter versus number of training timesteps on six sparse robotic environments: Maze Navigation (Row 1 Column 1), Pick and Place (Row 1 Column 2), Bin (Row 1 Column 3), Hollow (Row 2 Column 1), Rope Manipulation (Row 2 Column 2), and Franka Kitchen environment (Row 2 Column 3).

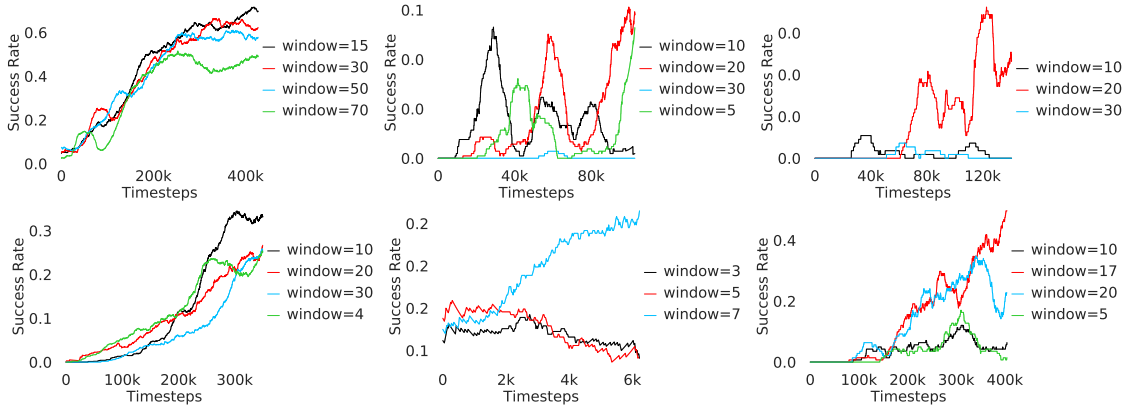


FIGURE 6.10: The success rate plots show the performance of RPL for values of k window size parameter versus number of training epochs on six sparse robotic environments: Maze Navigation (Row 1 Column 1), Pick and Place (Row 1 Column 2), Bin (Row 1 Column 3), Hollow (Row 2 Column 1), Rope Manipulation (Row 2 Column 2), and Franka Kitchen environment (Row 2 Column 3).

6.3 Related Work

Prior methods [131, 136, 137] leverage expert demonstrations to improve sample efficiency and accelerate learning, where some methods use imitation learning to bootstrap learning [69, 120, 138, 139]. Some approaches use fixed relabeling [89] for performing task

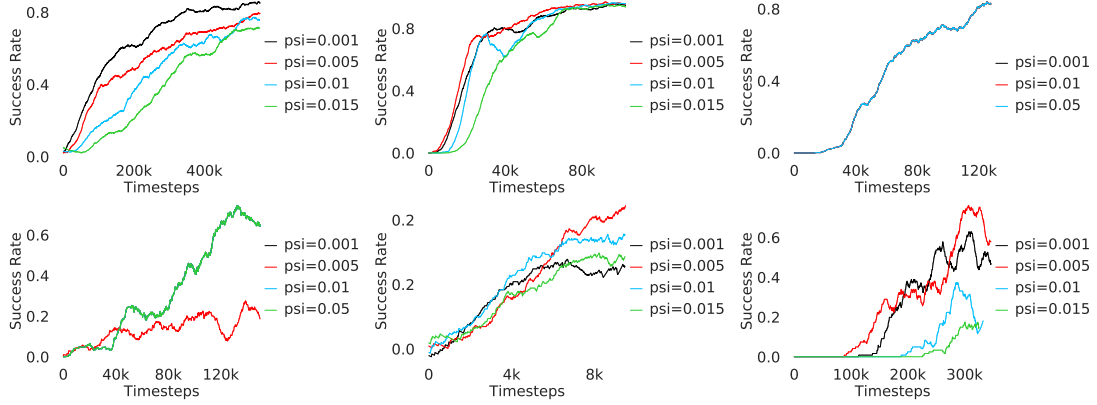


FIGURE 6.11: The success rate plots show performance of PEAR for values of learning weight parameter ψ versus number of training timesteps on six sparse robotic environments: Maze Navigation (Row 1 Column 1), Pick and Place (Row 1 Column 2), Bin (Row 1 Column 3), Hollow (Row 2 Column 1), Rope Manipulation (Row 2 Column 2), and Franka Kitchen environment (Row 2 Column 3).

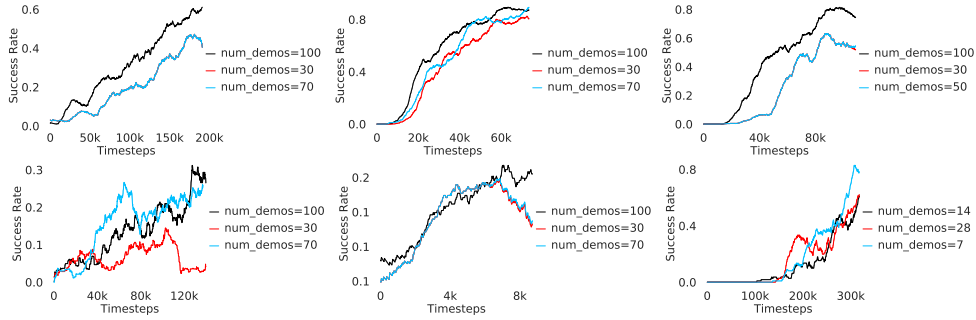


FIGURE 6.12: The success rate plots show success rate performance plots of varying number of expert demonstrations versus number of training epochs on six sparse robotic environments: Maze Navigation (Row 1 Column 1), Pick and Place (Row 1 Column 2), Bin (Row 1 Column 3), Hollow (Row 2 Column 1), Rope Manipulation (Row 2 Column 2), and Franka Kitchen environment (Row 2 Column 3).

segmentation. However, such approaches may cause unbalanced task split between hierarchical levels. In contrast, our approach sidesteps this limitation by properly balancing hierarchical levels using adaptive relabeling. Intuitively, we enable balanced task split, thereby avoiding degenerate solutions. Recent approaches restrict subgoal space using adjacency constraints [140], employ graph based approaches for decoupling task horizon [141], or incorporate imagined subgoals combined with KL-constrained policy iteration scheme [28]. However, such approaches assume additional environment constraints and only work on relatively shorter horizon tasks with limited complexity.

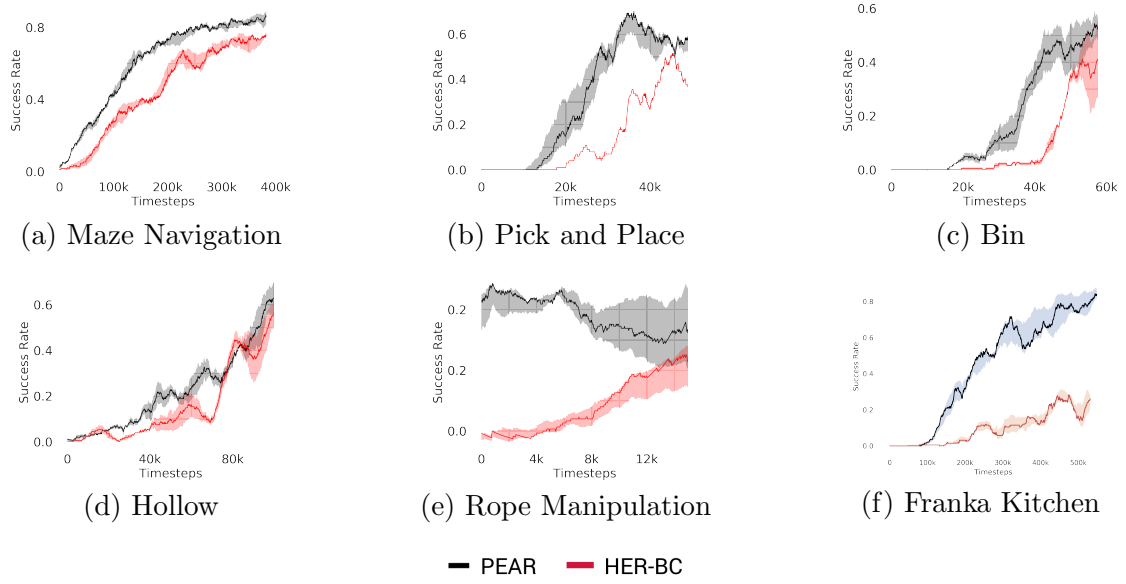


FIGURE 6.13: **Comparison with HER-BC baseline:** This figure depicts success rate plots of PEAR-IRL compared with the HER-BC baseline (a single-level implementation of Hindsight Experience Replay with expert demonstrations) on six sparse robotic environments: Maze Navigation (a), Pick and Place (b), Bin (c), Hollow (d), Rope Manipulation (e), and Franka Kitchen environment (f). PEAR consistently outperforms this baseline, demonstrating the advantages of our hierarchical formulation in complex tasks.

[142] is an inter-level cooperation based approach for generating achievable subgoals, However, the approach requires extensive exploration for selecting good subgoals, whereas our approach rapidly enables effective subgoal generation using primitive enabled adaptive relabeling. In order to accelerate RL, recent works firstly learn behavior skill priors [143, 144] from expert data or pre-train policies over a related task, and then fine-tune using RL. Such approaches largely depend on policies learnt during pre-training, and are hard to train when the source and target task distributions are dissimilar.

Other approaches either use bottleneck option discovery [145] or behavior priors [146, 147] to discover and embed behaviors from past experience, or directly hand-design action primitives [133, 148]. While this simplifies the higher level task, explicitly designing action primitives is tedious for hard tasks, and may lead to sub-optimal predictions. Since PEAR learns multi-level policies in parallel, the lower level policies can learn required optimal behavior, thus avoiding the issues with prior approaches.

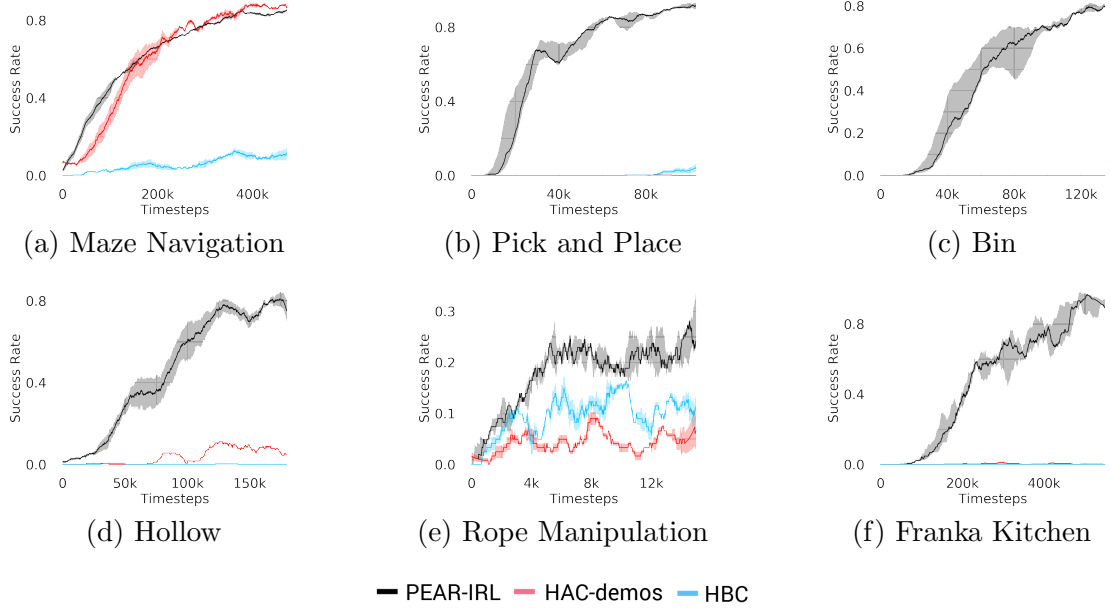


FIGURE 6.14: **Success rate comparison: PEAR-IRL vs HAC-demos vs HBC** This figure compares the success rate performances of PEAR-IRL with HAC-demos and HBC on six sparse robotic environments: Maze Navigation (a), Pick and Place (b), Bin (c), Hollow (d), Rope Manipulation (e), and Franka Kitchen environment (f). HAC-demos uses hierarchical actor-critic [134] as the RL objective and is jointly optimized using an additional behavior cloning objective, where the lower level uses primitive expert demonstrations and the upper level uses subgoal demonstrations extracted using a fixed window-based approach (as in RPL [89]). HBC (Hierarchical Behavior Cloning) uses the same demonstrations as HAC-demos at both levels but is trained using only the behavior cloning objective (thus, without RL). As seen in the figure, although HAC-demos shows good performance in the easier Maze Navigation environment, both HAC-demos and HBC fail to solve harder tasks, and PEAR-IRL significantly outperforms both baselines. The solid line and shaded region represent the mean and range of success rates across 5 seeds.

6.4 Discussion

Limitations In this work, we assume availability of directed expert demonstrations, which we plan to deal with in future. Additionally, D_g is periodically re-populated, which is an additional overhead and might be a bottleneck in tasks where relabeling cost is high. Notably, we side-step this limitation by passing the whole expert trajectory as a mini-batch for a single forward pass through lower primitive.

Conclusion and future work We propose primitive enabled adaptive relabeling (PEAR), a HRL and IL based approach that performs adaptive relabeling on a few expert demonstrations to solve complex long horizon tasks. We perform comparisons with a various

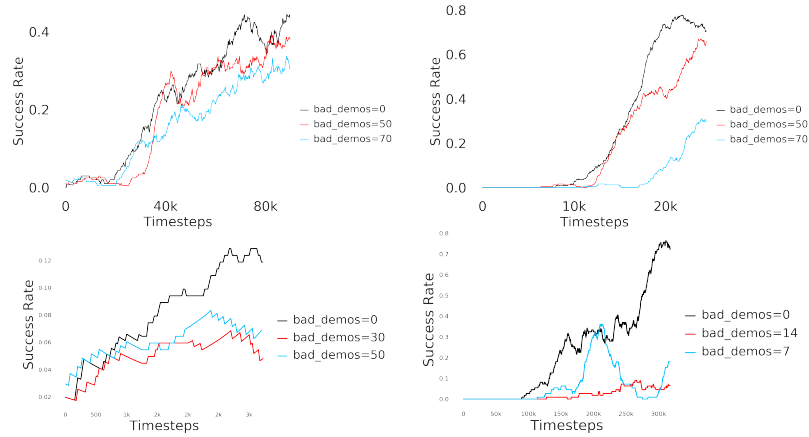


FIGURE 6.15: **Ablation with sub-optimal demonstrations:** The success rate plots show the performance of PEAR-IRL with varying number of sub-optimal demonstrations in the expert demonstration dataset on six sparse robotic environments: Maze Navigation (Row 1 Column 1), Pick and Place (Row 1 Column 2), Rope Manipulation (Row 2 Column 1), and Franka Kitchen environment (Row 2 Column 2). As can be seen, the performance suffers with increasing number of sub-optimal demonstrations.

baselines and demonstrate that **PEAR** shows strong results in simulation and real world robotic tasks. In future work, we plan to address harder sequential decision making tasks, and plan to analyse generalization beyond expert demonstrations. We hope that **PEAR** encourages future research in the area of adaptive relabeling and primitive informed regularization, and leads to efficient approaches to solve long horizon tasks.

Part III

Aligning Control with Preferences

Chapter 7

PIPER: Primitive-Informed Preference-based Hierarchical Reinforcement Learning via Hindsight Relabeling

We are our choices.

– Jean-Paul Sartre

In previous two chapters, we proposed two hierarchical approaches which are able to effectively leverage expert demonstrations for solving complex tasks. Since acquiring such expert demonstrations is non-trivial in harder tasks, we next ask the question: *can we still learn effective hierarchical strategies and solve the inherent issues of non-stationarity and infeasible subgoal generation without assuming access to expert demonstrations?*

¹ Recent approaches in preference-based learning [33, 149, 150] use human feedback to learn a reward model and subsequently apply RL to solve the task using the learned reward model. Impressively, such methods have demonstrated performance comparable

¹This chapter is based on the paper published previously at ICML 2024 [79].

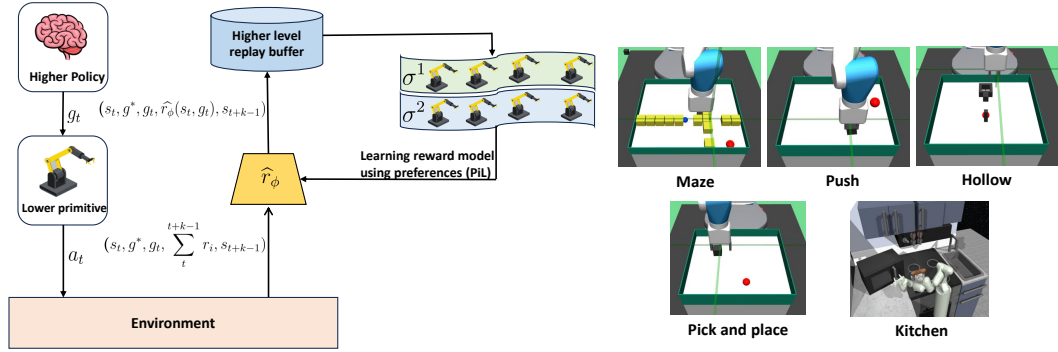


FIGURE 7.1: PIPER Overview

PIPER Overview This figure shows the overview of PIPER (left). The higher level policy predicts subgoals g_t for the lower primitive, which executes actions a_t on the environment. We propose to learn a preference-based reward model $\hat{r}_\phi$ using our PiL feedback on higher level trajectories sampled from higher level replay buffer, and subsequently use $\hat{r}_\phi$ to relabel the replay buffer transitions, thereby mitigating non-stationarity in HRL. On the right, we depict the training environments: (i) maze navigation environment, (ii) pick and place environment, (iii) push environment, (iv) hollow environment, and (v) franka kitchen environment.

to RL policies trained with access to ground truth rewards. This raises the following question: *can advances in preference-based learning be used to simultaneously address the twin issues of reward non-stationarity and infeasible subgoal generation in HRL?*

In this work, we provide an affirmative answer to the above question by proposing PIPER: **Primitive Informed Preference-based hierarchical reinforcement learning via hindsight Relabeling**. The key realization underlying PIPER is that a suitably designed preference-based RL approach can be used to learn a high-level reward model that is simultaneously decoupled from non-stationary, lower-level rewards and carefully tailored to generate feasible subgoals. To achieve this, several innovations are necessary. First, to overcome the problem of acquiring the human preference feedback needed to learn the higher-level reward model, we propose a goal-conditioned, sparse reward-based approach to replace human trajectory preferences. Second, to address the sample-inefficiency arising from the use of sparse rewards, we apply hindsight experience relabeling [95] to learn a denser, more informative higher-level reward model. Finally, to encourage our higher-level policies to predict subgoals achievable by lower-level policies, we propose a novel value-function

regularization scheme that calibrates subgoal selection to the current lower-level policy’s abilities. Taken together, these techniques form the core of PIPER (see Figure 7.1). We perform extensive experiments in complex, sparse-reward environments that empirically show that PIPER demonstrates impressive performance and consistently outperforms the baselines, as well as ablation studies illustrating the importance of each of PIPER’s component techniques.

We summarize the main contributions of PIPER:

- We demonstrate that PIPER is able to mitigate non-stationarity in off-policy HRL, by employing preference-based feedback.
- Since collecting human preference feedback is impractical, we propose an alternative primitive-in-the-loop (PiL) scheme to determine preferences between trajectories without human feedback.
- Using primitive informed regularization, PIPER generates efficient subgoals, thus improving performance.
- We employ hindsight relabeling to improve sample efficiency in preference-based learning and deal with sparsity in sparse-reward scenarios.
- PIPER uses soft target updates to mitigate instability due to non-stationary reward model learned using preference-based feedback.
- PIPER achieves greater than 50% success rates in challenging, sparse-reward robotic environments, where most other hierarchical and non-hierarchical baselines fail to achieve any significant progress.

7.0.1 Limitations of Existing Approaches to HRL

While HRL promises significant advantages over non-hierarchical RL, including improved sample efficiency due to temporal abstraction and improved exploration [6, 26], serious challenges remain. In this work, we focus on two outstanding issues:

C1: the destabilizing effect of lower-level reward non-stationarity on off-policy HRL,

C2: and performance degradation due to infeasible subgoal generation by higher-level policies.

As discussed in [26] and [27], off-policy HRL suffers from non-stationarity due to changing lower primitive behavior, since, for a given transition in the higher-level replay buffer $(s_t, g^*, g_t, \sum_{i=t}^{t+k-1} r_i^H, s_{t+k})$, the rewards and transition may have been generated by an outdated lower-level policy. In addition, it was observed in [28] that, without taking care to ensure valid subgoal selection at the higher level, HRL methods may issue unachievable subgoals to the lower-level policy, stalling learning at the lower level and significantly impacting performance. The primary motivation of this work is the development of a novel, preference-based learning-inspired technique for learning the high-level reward function, r^H , that addresses these key limitations of existing HRL methods.

7.1 Primitive Informed Preference Based Learning

We consider an MDP $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $p : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$ is the transition probability function mapping state-action pairs to probability distributions over the state space, $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function, and $\gamma \in (0, 1)$ is the discount factor. At a given timestep t , the agent is in state s_t , takes action $a_t \sim \pi(\cdot|s_t)$ according to some policy $\pi : \mathcal{S} \rightarrow \Delta(\mathcal{A})$ which maps states to probability distributions over the action space, receives reward $r_t = r(s_t, a_t)$, and the system transitions to a new state $s_{t+1} \sim p(\cdot|s_t, a_t)$. The standard reinforcement learning (RL) objective is given by

$$\pi^* := \arg \max_{\pi} J(\pi) = \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right], \quad (7.1)$$

which is also called policy optimization in literature. The state action value function $Q(s, g, a_i)$ computes the expected cumulative reward when the start state is s , g is the final goal, and the next primitive action is a_i . In hierarchical RL (HRL) problem, the

higher-level policy aims to achieve an end goal by issuing subgoals to the lower-level policy, while the lower-level policy chooses primitive actions oriented towards achieving the specified subgoals.

In HRL, the higher-level policy $\pi^H : \mathcal{S} \rightarrow \Delta(\mathcal{G})$ specifies a subgoal $g_t \in \mathcal{G}$, where $\mathcal{G} \subset \mathcal{S}$ is the set of possible goals. During execution at each time step t , the subgoal $g_t \sim \pi^H(\cdot|s_t)$ after every k timesteps and $g_t = g_{k \cdot \lceil t/k \rceil}$, otherwise. The effect of this is that the higher-level policy issues new subgoals every k timesteps and keeps subgoals fixed in between.

Furthermore, at each t , the lower-level policy $\pi^L : \mathcal{S} \times \mathcal{G} \rightarrow \Delta(\mathcal{A})$ selects primitive actions $a_t \sim \pi^L(\cdot|s_t, g_t)$ according to the current state and subgoal specified by π^H , and the state transitions to $s_{t+1} \sim p(\cdot|s_t, a_t)$. Finally, at each timestep t , the higher level of the hierarchy provides the lower level with reward $r_t^L = r^L(s_t, g_t, a_t) = -\mathbf{1}_{\{\|s_t - g_t\|_2 > \varepsilon\}}$, where $\mathbf{1}_B$ is the indicator function on a given set B , while the higher level receives reward $r_t^H = r^H(s_t, g^*, g_t)$, where $g^* \in \mathcal{G}$ is the end goal and $r^H : \mathcal{S} \times \mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R}$ is a high-level reward function that we have yet to specify. The lower level populates its replay buffer with samples $(s_t, g_t, a_t, r_t^L, s_{t+1})$, while, at each t such that after every k timesteps, the higher level populates its buffer with samples of the form $(s_t, g^*, g_t, \sum_{i=t}^{t+k-1} r_i^H, s_{t+k})$. Next, we highlight key limitations of existing HRL methods.

We introduce our approach, PIPER: **P**rimitive **I**nformed **P**rEference-based hierarchical reinforcement learning via hindsight **R**elabeling for solving complex sparse-reward tasks. The motivating idea behind our approach is that preference-based RL, where human preferences are used to learn a dense, informative reward function, can be used to learn a high-level reward function that simultaneously mitigates the issues of reward non-stationarity and infeasible subgoal generation described in Section 7.0.1. Before presenting the main idea, let us briefly discuss preference-based learning.

7.1.1 Preference-based Learning (PBL)

In traditional RL, agents learn to maximize the accumulated rewards $r(s_t, a_t)$ obtained from the environment, where the reward $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is assumed to be known. In

many real-world scenarios, however, suitable environment rewards are notoriously difficult to construct and typically require domain-specific knowledge [151]. To learn the high-level reward necessary for the HRL setting, we use a preference-based RL setup, where the agent learns to perform the high-level task using preferences over agent behaviors [33, 149, 150, 152].

In the preference-based setting, agent behavior over a k -length trajectory can be represented by a sequence, δ , of state observations and actions: $\delta = ((s_t, a_t), (s_{t+1}, a_{t+1}) \dots (s_{t+k-1}, a_{t+k-1}))$. The goal is to learn a reward function, $\hat{r}_\phi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, with neural network parameters ϕ , such that preferences between any two trajectories δ^1, δ^2 can be modeled using the Bradley-Terry model [153]:

$$P_\phi [\delta^1 \succ \delta^2] = \frac{\exp \sum_t \hat{r}_\phi (s_t^1, a_t^1)}{\sum_{i \in \{1,2\}} \exp \sum_t \hat{r}_\phi (s_t^i, a_t^i)}, \quad (7.2)$$

where $\delta^1 \succ \delta^2$ denotes the event that δ^1 is preferred over δ^2 . To learn parameters ϕ such that (7.2) matches the true preferences, behavior and preference data are recorded in a dataset $\mathcal{D}$ with entries of the form (δ^1, δ^2, y) , where $y = (1, 0)$ when δ^1 is preferred over δ^2 , $y = (0, 1)$ when δ^2 is preferred over δ^1 , and $y = (0.5, 0.5)$ when there is no preference. The standard approach in the preference-based literature (see [33, 150]), which we adopt in PIPER, is to learn the reward function $\hat{r}_\phi$ using a cross-entropy loss:

$$L(\phi) = - \sum_{\mathcal{D}} (y_1 \log P_\phi [\delta^1 \succ \delta^2] + y_2 \log P_\phi [\delta^2 \succ \delta^1]), \quad (7.3)$$

where $(\delta^1, \delta^2, y) \in \mathcal{D}$ and y_1 and y_2 denote the first and second entries of y . Given the preference model and learning objective in (7.2) and (7.3), we now turn to the problem of acquiring preference data in our hierarchical setting.

Challenges of directly applying PBL in HRL:

- First, collecting the quantity of human feedback needed to enable preference-based learning of the high-level reward function is impractical. To address this, we propose

a replacement PiL scheme to determine preferences between trajectories without human feedback.

- Second, standard goal-conditioned rewards are too sparse for sample-efficient learning. We therefore incorporate hindsight relabeling to reduce sparsity and increase the informativeness of observed trajectories.
- Third, even with hindsight relabeling the higher-level policy may choose subgoals that are infeasible for the lower-level policy, so we regularize the PiL reward with the value function of the lower-level policy to encourage feasible subgoal selection.
- Finally, we incorporate soft target updates [154] to mitigate the training instability while learning the high-level reward function. Taken together, this combination of techniques constitutes PIPER. The rest of this section provides details on each aspect of our approach. Pseudocode for PIPER is provided in Algorithm 5.

7.1.2 PiL: Primitive-in-the-Loop Feedback for HRL

We next introduce our Primitive-in-the-Loop (PiL) approach for generating higher-level preference feedback. This technique provides an effective, easily computable replacement for human feedback, avoiding the impracticality of obtaining human preference data for trajectories observed during training in the HRL setting. The key idea is to replace human preferences over higher-level trajectories with preferences generated by using a simple, g^* -conditioned sparse reward to perform pairwise comparisons. We note that the sparsity of the proposed reward presents additional issues, which we address in Section 7.1.3 below.

Implicit reward functions: Let s be the current state, g be the subgoal predicted by higher level, and g^* be the final goal. In our goal-conditioned HRL setting, we represent k -length trajectories of lower-level primitive behavior as sequences Δ given by $\Delta = ((s_t, a_t), (s_{t+1}, a_{t+1}), \dots, (s_{t+k-1}, a_{t+k-1}))$, and n -length trajectories of higher-level behavior as sequences σ of states and subgoal predictions, i.e.,

$\sigma = ((s_t, g_t), (s_{t+k}, g_{t+k}), \dots, (s_{t+(n-1)k}, g_{t+(n-1)k}))$, where the higher-level policy is executed for n timesteps and the lower-level policy is executed for k timesteps between each pair of consecutive subgoal predictions. In preference-based learning, the preferences are assumed to correspond to an implicit reward function $r : \mathcal{S} \times \mathcal{G} \times \mathcal{A} \rightarrow \mathbb{R}$ implied by the true preferences. Let g^* be the final goal and σ^1 and σ^2 be two higher-level trajectories. Whenever σ^1 is preferred to σ^2 , denoted by $\sigma^1 \succ \sigma^2$, then the assumption that r encodes the true preferences implies:

$$\sum_{i=0}^{n-1} r(s_{t+ik}^1, g^*, g_{t+ik}^1) > \sum_{i=0}^{n-1} r(s_{t+ik}^2, g^*, g_{t+ik}^2) \quad (7.4)$$

Replacing human feedback: In the standard preference-learning framework, preferences are elicited from human feedback [33] and are subsequently used to learn the reward model $\hat{r}_\phi$. In this work, we replace this Human-in-the-Loop (HiL) feedback with Primitive-in-the-Loop (PiL) feedback by using implicit sparse rewards, $r^s(s_t, g^*, g_t)$, defined presently, to determine preferences y between behavior sequences σ^1 and σ^2 . We call this feedback *Primitive-in-the-Loop* since we generate this feedback using primitive sparse rewards. We obtain these primitive rewards as follows. Suppose the higher level policy π^H predicts subgoal $g_t \sim \pi^H(\cdot | s_t, g^*)$ for state s_t and goal g^* . The lower-level primitive executes primitive actions according to its policy π^L for k timesteps and ends up in state s_{t+k-1} . We use the sparse reward provided by the environment at state s_{t+k-1} as the implicit sparse reward, i.e., $r^s(s_t, g^*, g_t) = -\mathbf{1}_{\{\|s_{t+k-1} - g^*\|_2 > \epsilon\}}$. Note that this reward is directly available from the environment. We replace the implicit reward r in Equation (7.4) with r^s , and thus use r^s to obtain preferences between higher level behavior sequences. Concretely, for the goal g^* , $\sigma^1 \succ \sigma^2$ implies

$$\sum_{i=0}^{n-1} r^s(s_{t+ik}^1, g^*, g_{t+ik}^1) > \sum_{i=0}^{n-1} r^s(s_{t+ik}^2, g^*, g_{t+ik}^2) \quad (7.5)$$

The preferences elicited using r^s are subsequently used to learn the preference reward function $\hat{r}_\phi$.

Effect on non-stationarity: Off-policy HRL approaches suffer from non-stationarity due to outdated transitions in the higher-level replay buffer caused by the changing lower-level policy. In our approach, the rewards in the higher-level replay buffer are re-labeled using $\hat{r}_\phi$. Hence, the transitions are updated from $(s_t, g^*, g_t, \sum_{i=t}^{t+k-1} r_i, s_{t+k-1})$ to $(s_t, g^*, g_t, \hat{r}_\phi(s_t, g_t), s_{t+k-1})$. Since $\hat{r}_\phi$ does not depend on changing lower primitive behavior, this reward relabeling eliminates the non-stationarity typically encountered in off-policy HRL.

7.1.3 PiL with Goal-conditioned Hindsight Relabeling

Despite the intuitive appeal of the sparse PiL feedback reward r^s proposed in Section 7.1.2 for providing preferences between higher-level trajectories, due to its sparsity it mostly fails to generate a meaningful reward signal. As a result, learning an appropriate reward function $\hat{r}_\phi$ using r^s as our PiL primitive is unreliable. To address this issue, we employ hindsight relabeling [95] when comparing trajectories σ^1 and σ^2 . Specifically, we first randomly sample a new goal $\hat{g}$ from the set $\{s_{t+ik}^1, s_{t+ik}^2\}_{i=1}^{n-1}$ of states encountered during trajectories σ^1, σ^2 , then apply Equation (7.5) with g^* replaced by $\hat{g}$:

$$\sum_{i=0}^{n-1} r^s(s_{t+ik}^1, \hat{g}, g_{t+ik}^1) > \sum_{i=0}^{n-1} r^s(s_{t+ik}^2, \hat{g}, g_{t+ik}^2). \quad (7.6)$$

Using hindsight relabeling, we are able to generate significantly better preference feedback, resulting in improved sample efficiency and performance during training. We show in Section 7.2, Figure 7.4 that this simple hindsight relabeling approach significantly boosts performance in complex sparse-reward tasks.

7.1.4 Primitive-informed Regularization

As discussed above, the preference reward learned using PiL with hindsight relabeling motivates the higher-level policy to reach the goal g , while mitigating non-stationarity. However, the higher-level subgoal predictions g_t may be too difficult for the current lower-level policy, which may stall learning at the lower level (see Figure 7.2 for a comparison of PIPER with and without regularization). Ideally, the higher-level policy should produce subgoals at an appropriate level of difficulty, according to the current capabilities of the lower primitive. Properly balancing the task split between hierarchical levels is a recurring challenge in HRL.

For a given lower-level policy $\pi^L : \mathcal{S} \times \mathcal{G} \rightarrow \mathcal{A}$, denote the corresponding state value function by $V_{\pi^L}(s, g)$. To encourage appropriate subgoal selection by the higher-level policy, we propose using the lower-level state value function V_{π^L} to regularize the higher-level policy to predict feasible subgoals for the lower-level policy. Intuitively, $V_{\pi^L}(s_t, g_t)$ provides an estimate of the achievability of subgoal g_t from current state s_t , since a high value of $V_{\pi^L}(s_t, g_t)$ implies that the lower level expects to achieve high reward for subgoal g_t . Let $r^s(s, g^*, g)$ denote the parameterized reward model corresponding to the preference data, as defined in Section 7.1.2. For a trajectory τ of length T , consider the following KL-regularized formulation of preference-based learning, where π_{reg} is the regularizing policy for the higher-level policy π^H :

$$\max_{\pi^H} \mathbb{E}_{\pi^H} \left[\sum_{t=0}^T (r^s(s, g^*, g) - \beta \mathbb{D}_{\text{KL}}[\pi^H(\cdot | s_t) \| \pi_{reg}(\cdot | s_t)]) \right], \quad (7.7)$$

where $\beta \geq 0$ is a scalar hyperparameter controlling the deviation from the regularization policy π_{reg} . We propose the following formulation of the regularization policy:

$$\pi_{reg}(g_t | s_t) = \frac{\exp(m(V_{\pi^L}(s_t, g_t)))}{\hat{Z}(s_t)}, \quad (7.8)$$

where $\hat{Z}(s_t) = \sum_{g_t} \exp(m(V_{\pi^L}(s_t, g_t)))$, and $m = \frac{\alpha}{\beta}$. The policy $\pi_{reg}(\cdot | g_t)$ is simply the softmax distribution generating a given subgoal g_t with probability proportional to its

value $V_{\pi^L}(s_t, g_t)$. We substitute (7.8) in (7.7) to get

$$\max_{\pi^H} \mathbb{E}_{\pi^H} \left[\sum_{t=0}^T (r^s(s, g^*, g) + \alpha(V_{\pi^L}(s_t, g_t)) + \hat{m}(s_t)) \right], \quad (7.9)$$

where $\hat{m}(s_t) = \beta \mathcal{H}(s_t) - \beta \log Z(s_t)$, and $\mathcal{H}(s_t) = -\log \pi^H(g_t|s_t)$ is the entropy term for π^H . Following prior work [155, 156], we get the following optimal solution for the higher-level policy:

$$\pi^H(g_t|s_t) = \frac{1}{Z(s_t)} \exp\left(\frac{1}{\beta}(r^s(s, g^*, g) + \alpha(V_{\pi^L}(s_t, g_t)))\right), \quad (7.10)$$

where $Z(s_t) = \sum_{g_t} \exp(\frac{1}{\beta}(r^s(s, g^*, g) + \alpha(V_{\pi^L}(s_t, g_t))))$ is the partition function and α is the primitive regularization weight hyperparameter. We provide the complete derivation below (7.1.4.1). Notice that this optimal policy $\pi^H(g_t|s_t)$ assigns high probability to subgoals g_t which maximize the regularized reward $r^{total}(s, g^*, g) = r^s(s, g^*, g) + \alpha(V_{\pi^L}(s_t, g_t))$.

7.1.4.1 Deriving the final optimum of KL-Constrained Reward Maximization Objective

In this appendix, we will derive Eqn 7.10 from Eqn 7.7. Thus, we optimize the following objective:

$$\max_{\pi_U} \mathbb{E}_{\pi_U} \left[\sum_{t=0}^T (r^s(s, g^*, g) - \beta \mathbb{D}_{\text{KL}}[\pi^H(g_t|s_t) \parallel \pi_{reg}(g_t|s_t)]) \right]. \quad (7.11)$$

Re-writing the above equation after expanding KL divergence:

$$= \max_{\pi_U} \mathbb{E}_{\pi_U} \left[\sum_{t=0}^T (r^s(s, g^*, g) - \beta \log \frac{\pi^H(g_t|s_t)}{\pi_{reg}(g_t|s_t)}) \right] \quad (7.12)$$

$$= \max_{\pi_U} \mathbb{E}_{\pi_U} \left[\sum_{t=0}^T (r^s(s, g^*, g) - \beta \log \pi^H(g_t|s_t) + \beta \log \pi_{reg}(g_t|s_t)) \right]. \quad (7.13)$$

Here, we substitute π_{reg} from Eqn 7.8, and $m = \frac{\alpha}{\beta}$ in Equation 7.13,

$$= \max_{\pi_U} \mathbb{E}_{\pi_U} \left[\sum_{t=0}^T (r^s(s, g^*, g) - \beta \log \pi^H(g_t|s_t) + \beta \log \exp(m(V_{\pi^L}(s_t, g_t))) - \beta \log \sum_{g_t} \exp(m(V_{\pi^L}(s_t, g_t)))) \right] \quad (7.14)$$

$$= \max_{\pi_U} \mathbb{E}_{\pi_U} \left[\sum_{t=0}^T (r^s(s, g^*, g) - \beta \log \pi^H(g_t|s_t) + \alpha(V_{\pi^L}(s_t, g_t)) - \beta \log \sum_{g_t} \exp(m(V_{\pi^L}(s_t, g_t)))) \right] \quad (7.15)$$

$$= \min_{\pi_U} \mathbb{E}_{\pi_U} \left[\sum_{t=0}^T (\log \pi^H(g_t|s_t) - \frac{1}{\beta} (r^s(s, g^*, g) + \alpha(V_{\pi^L}(s_t, g_t))) + \log \sum_{g_t} \exp(m(V_{\pi^L}(s_t, g_t)))) \right] \quad (7.16)$$

$$= \min_{\pi_U} \mathbb{E}_{\pi_U} \left[\sum_{t=0}^T (\log \left(\frac{\pi^H(g_t|s_t)}{\exp(\frac{1}{\beta} (r^s(s, g^*, g) + \alpha(V_{\pi^L}(s_t, g_t)))} \right) + \log \sum_{g_t} \exp(m(V_{\pi^L}(s_t, g_t)))) \right] \quad (7.17)$$

$$\begin{aligned}
&= \min_{\pi_U} \mathbb{E}_{\pi_U} \left[\sum_{t=0}^T \left(\log \left(\frac{\pi^H(g_t|s_t)}{\frac{1}{Z(s_t)} \exp(\frac{1}{\beta}(r^s(s, g^*, g) + \alpha(V_{\pi^L}(s_t, g_t))))} \right) \right. \right. \\
&\quad \left. \left. + \log \sum_{g_t} \exp(m(V_{\pi^L}(s_t, g_t))) - \log Z(s_t) \right) \right], \tag{7.18}
\end{aligned}$$

where, $Z(s_t) = \sum_{g_t} \exp(\frac{1}{\beta}(r^s(s, g^*, g) + \alpha(V_{\pi^L}(s_t, g_t))))$ is the partition function.

Note that the partition function $Z(s_t)$ and the term $\log \sum_{g_t} \exp(m(V_{\pi^L}(s_t, g_t)))$ do not depend on policy π_U . We define

$$\pi^{H^*}(g_t|s_t) = \frac{1}{Z(s_t)} \exp(\frac{1}{\beta}(r^s(s, g^*, g) + \alpha(V_{\pi^L}(s_t, g_t)))), \tag{7.19}$$

which is a valid probability distribution, as $\pi^{H^*}(g_t|s_t) \geq 0$ and $\sum_{g_t} \pi^{H^*}(g_t|s_t) = 1$. We can re-organize the equation as

$$\begin{aligned}
&= \min_{\pi_U} \mathbb{E}_{\pi_U} \left[\sum_{t=0}^T (\mathbb{D}_{\text{KL}}[\pi^H(g_t|s_t) \parallel \pi^{H^*}(g_t|s_t)] - \log \sum_{g_t} \exp(m(V_{\pi^L}(s_t, g_t))) - \log Z(s_t)) \right], \tag{7.20}
\end{aligned}$$

where $\pi^{H^*}(g_t|s_t)$ is minimized when the KL-divergence is 0, according to Gibbs' inequality.

Hence, we get the optimal solution

$$\pi^H(g_t|s_t) = \pi^{H^*}(g_t|s_t) = \frac{1}{Z(s_t)} \exp(\frac{1}{\beta}(r^s(s, g^*, g) + \alpha(V_{\pi^L}(s_t, g_t)))). \tag{7.21}$$

If we use r^{total} to generate preferences between trajectories instead of the standard preferences outlined in Section 7.1.1, we end up with the optimal policy under this primitive-regularized, preference-based learning scheme. Hence, we substitute $r^{total}(s, g^*, g)$ into inequality (7.6) to yield our final preference condition for determining if $\delta^1 \succ \delta^2$:

$$\sum_{i=0}^{n-1} r^{total}(s_{t+ik}^1, \hat{g}, g_{t+ik}^1) > \sum_{i=0}^{n-1} r^{total}(s_{t+ik}^2, \hat{g}, g_{t+ik}^2). \quad (7.22)$$

PIPER uses the preferences elicited by (7.22) to learn the reward function $\hat{r}_\phi$, which is in turn used to perform reward relabeling of the higher-level replay buffer. As illustrated in Figure 7.2, value regularization can lead to significantly improved performance in certain tasks, while leading to minimal performance degradation in others.

7.1.5 PIPER Implementation

Reward stabilization using target networks: To mitigate potential training instability when the reward model $\hat{r}_\phi$ is learned using preference-based learning, we utilize target networks with soft target updates [154]. In practice, we found this to greatly stabilize learning.

Pseudocode details: We explain our approach in detail in Algorithm 5. The higher- and lower-level policies are both trained using Soft Actor Critic (SAC) [114]. The rewards in higher level transitions are relabeled using the reward model $\hat{r}_\phi$. Instead of relabeling all replay buffer transitions, we relabel the higher-level replay buffer transitions as they are sampled while training.

7.2 Experiments

We perform experiments to empirically investigate the following questions:

- (1) How well does PIPER perform in sparse maze navigation and robotic manipulation tasks?
- (2) Is PIPER able to mitigate the recurring issue of non-stationarity in HRL?
- (3)

Algorithm 5 PIPER

```

1: Initialize preference dataset  $D = \{\}$ 
2: Initialize higher level replay buffer  $\mathcal{R}^H = \{\}$  and lower level replay buffer  $\mathcal{R}^L = \{\}$ 
3: Initialize reward model parameters  $\phi$ 
4: Initialize target reward model parameters of  $\phi' \leftarrow \phi$ 
5: for  $i = 1 \dots N$  do
6:   // Collect experience using  $\pi^H$  and  $\pi^L$  and store transitions in  $\mathcal{R}^H$  and  $\mathcal{R}^L$ 
7:   for each timestep  $t$  do
8:      $d^H \leftarrow d^H \cup \{(s_t, g^*, g_t, \sum_{i=t}^{t+k-1} r_i, s_{t+k-1})\}$ 
9:      $d^L \leftarrow d^L \cup \{(s_t, g_t, a_t, r_t, s_{t+1})\}$ 
10:     $\mathcal{R}^H \leftarrow \mathcal{R}^H \cup d^H$ 
11:     $\mathcal{R}^L \leftarrow \mathcal{R}^L \cup d^L$ 
12:    // Sample higher level behavior trajectories and goal
13:     $(\sigma^1, \sigma^2) \sim \mathcal{R}^H$  and  $g \sim \mathcal{G}$ 
14:    Sample a set of additional goals for relabeling  $\mathcal{G}'$ 
15:    for  $g' \in \mathcal{G}'$  do
16:      Relabel  $g$  by  $g'$  and generate label  $y$  using Equation (7.22)
17:      Store preference  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\sigma^1, \sigma^2, y)\}$ 
18:    // Reward Model Learning
19:    for each gradient step do
20:      Optimize model reward  $\hat{r}_\phi$  using Equation (7.3)
21:      // Soft update target reward model parameters
22:       $\phi' \leftarrow \tau \phi + (1 - \tau) \phi'$ 
23:    // Policy Learning
24:    for each gradient step do
25:      Sample  $\{(\sigma_j)\}_{j=1}^m$  from  $\mathcal{R}^H$ 
26:      Sample  $\{(\delta_j)\}_{j=1}^m$  from  $\mathcal{R}^L$ 
27:      Relabel rewards in  $\{(\sigma_j)\}_{j=1}^m$  using target reward model  $\hat{r}_{\phi'}$ 
28:      Optimize higher policy  $\pi^H$  using SAC
29:      Optimize lower policy  $\pi^L$  using SAC

```

Does PIPER outperform flat preference-based learning? **(4)** Does PIPER enhance sample efficiency and training stability? **(5)** What are the qualitative insights into PIPER’s efficacy over baselines?

(6) What is the contribution of each of the design choices in PIPER?

7.2.1 Setup

We evaluate PIPER on four robotic navigation and manipulation tasks: (i) maze navigation, (ii) pick and place [95], (iii) push, (iv) hollow, and (v) franka kitchen [89]. Since we

focus on sparse reward scenarios, we re-implement these environments as sparse reward environments. Notably, since the pick and place, push, hollow and kitchen task environments are difficult, in order to speedup training, we assume access to a single human demonstration, and use an additional imitation learning objective at the lower level. We do not assume access to any demonstration in the maze navigation task. We keep this assumption consistent among all baselines to ascertain fair comparisons.

For environments (i) – (v) ((i) maze navigation, (ii) pick and place [95], (iii) push, (iv) hollow, and (v) franka kitchen [89]), the maximum task horizon $\mathcal{T}$ is set to 225, 50, 50, 100, 225 timesteps, respectively, and the lower primitive is allowed to execute for $k = 15, 7, 7, 10$ and 15 timesteps, respectively. For environments (i) – (v), the primitive regularization weight hyper-parameter α is set as $10^{-5}, 10^{-4}, 10^{-4}, 10^{-4}, 10^{-6}$ in environments (i) – (v), respectively (see Figure ??). In our experiments, we use off-policy Soft Actor Critic (SAC) [114] for optimizing RL objective, using the Adam [157] optimizer. The actor and critic networks are formulated as three-layer, fully connected neural networks with 512 neurons in each layer. The experiments are run for 6.75e5, 1.5e5, 6.5E5, and 6.75e5 timesteps in environments (i)–(iv), respectively. In our experiments, we use off-policy Soft Actor Critic (SAC) [114] for optimizing RL objective, using the Adam [157] optimizer. The actor and critic networks are formulated as three-layer, fully connected neural networks with 512 neurons in each layer. The experiments are run for 6.75e5, 1.5e5, 6.5E5, and 6.75e5 timesteps in environments (i) – (iv), respectively.

To ensure fair comparisons, we keep parameters including neural network layer width, number of layers, choice of optimizer, SAC implementation parameters, etc., consistent across all baselines. For implementing the RAPS baseline, we use the following lower-level behaviors: in maze navigation, we design a single primitive, *reach*, where the lower-level primitive travels in a straight line towards the subgoal predicted by higher level. In the pick and place, push and hollow tasks, we design three primitives: *gripper-reach*, where the gripper goes to given position (x_i, y_i, z_i) ; *gripper-open*, which opens the gripper; and *gripper-close*, which closes the gripper. In the kitchen environment, we use the action primitives implemented in RAPS [133].

7.2.1.1 Additional hyper-parameters

Here, we enlist the additional hyper-parameters used in PIPER:

Hyperparameter	Value	Description
activation	tanh	Activation for reward model
layers	3	Number of layers in the critic/actor networks
hidden	512	Number of neurons in each hidden layer
Q_lr	0.001	Critic learning rate
pi_lr	0.001	Actor learning rate
buffer_size	int(1E7)	Size of experience replay buffer
tau	0.8	Polyak averaging coefficient
clip_obs	200	Clip observation
n_cycles	1	Number of cycles per epoch
n_batches	10	Training batches per cycle
batch_size	1024	Batch size hyper-parameter
reward_batch_size	50	Reward batch size for PEBBLE and RFLAT
random_eps	0.2	Percentage of time a random action is taken
alpha	0.05	Weightage parameter for SAC
noise_eps	0.05	Std of Gaussian noise added to not-completely-random actions
norm_eps	0.01	Epsilon used for observation normalization
norm_clip	5	Normalized observations are cropped to this value
adam_beta1	0.9	Beta 1 for Adam optimizer
adam_beta2	0.999	Beta 2 for Adam optimizer

TABLE 7.1: Hyperparameters used in PIPER.

7.2.2 Evaluation and Results

In order to analyse our design choices, we consider multiple baselines: PIPER-No-V (PIPER without primitive informed regularization, Section 7.1.4), RFLAT (Single-level PEBBLE [33] implementation with target networks), HIER (vanilla hierarchical SAC implementation), HAC (Hindsight Actor Critic [27]), DAC (Discriminator Actor Critic [110]), FLAT (Single-level SAC), and RAPS [133]. In Figure 7.2, we compare the success rate performances of PIPER with hierarchical and non-hierarchical baselines.

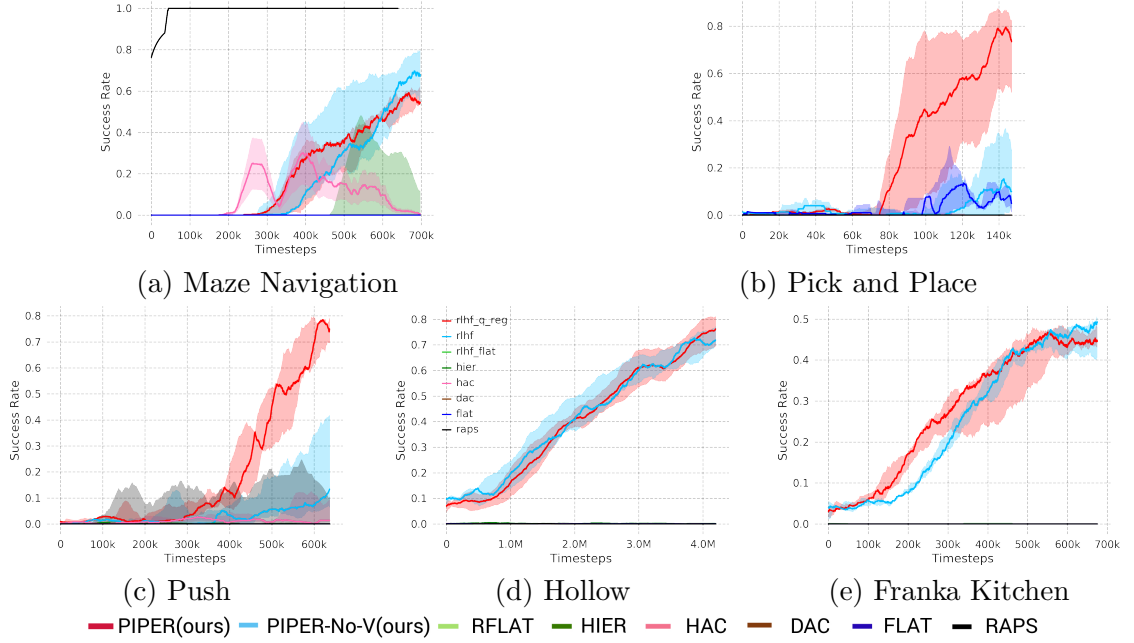


FIGURE 7.2: **Success rate comparison** This figure compares the success rate performances on four sparse maze navigation and robotic manipulation environments: (i) maze navigation (Row 1, Col 1), (ii) pick and place (Row 1, Col 2), (iii) push (Row 2, Col 1), (iv) hollow (Row 2, Col 2), and (v) franka kitchen (Row 2, Col 3). The solid line and shaded regions represent the mean and standard deviation, across 5 seeds. We compare our approach PIPER against multiple baselines. As can be seen, PIPER shows impressive performance and significantly outperforms the baselines.

Evaluating the Benefits of Primitive-Informed Regularization. To illustrate the advantage of primitive informed regularization, we implemented PIPER-No-V, which is PIPER without primitive-informed regularization. As seen in Figure 7.2, though PIPER significantly outperforms PIPER-No-V baseline in pick and place and push tasks, it only slightly outperforms PIPER-No-V in the maze, hollow and kitchen environments. This illustrates that primitive-informed regularization successfully encourages the higher-level policy to generate subgoals achievable by lower-level policies.

Comparison with hierarchical baselines. We compare PIPER with two hierarchical baselines to demonstrate that PIPER is indeed able to effectively mitigate non-stationarity in HRL. We implemented two baselines: HAC (Hierarchical Actor Critic) [95], a hierarchical approach that deals with the non-stationarity issue by relabeling transitions, while assuming an optimal lower primitive; and HIER, which is a vanilla HRL baseline implemented using SAC. As seen in Figure 7.2, PIPER is able to significantly outperform both

these baselines, illustrating that PIPER is indeed able to better mitigate non-stationarity in HRL. Although HAC addresses non-stationarity by considering optimal lower level policy, it is still not able to solve non-stationarity issue when this assumption does not hold. In contrast, PIPER employs preference based learning to generate efficient subgoals for the lower primitive, thus generating efficient subgoals for the lower primitive and mitigating non-stationarity in HRL.

Comparison with PEBBLE ablation. We also compare PIPER with RFLAT, which is a re-implementation of PEBBLE [33] with target networks, and where the preference feedback is generated using PiL approach. Since the environments are sparse, we also augmented PEBBLE with hindsight relabeling in our implementation of RFLAT. As seen in Figure 7.2, PIPER is able to significantly outperform RFLAT. This empirically illustrates that our hierarchical preference-based learning approach is able to outperform flat preference-based learning approach, thus demonstrating the advantage of hierarchical abstraction.

Comparison against RAPS. We finally compare against RAPS [133], a hierarchical approach where the higher-level policy picks from among hand-designed action primitives at the lower level. Importantly, the performance of RAPS depends on the quality of action primitives. We found that RAPS was able to outperform PIPER and all other baselines in the maze navigation environment. We hypothesize that this is because the hand-designed lower primitive used in the maze task makes the task significantly easier for the higher level. However, RAPS did not show progress in the more difficult pick and place, push, hollow and kitchen tasks. This demonstrates that although RAPS takes advantage of domain-specific action primitives, the performance does not generalize to complex environments due to sub-optimal action primitives. In contrast, PIPER does not require domain-specific action primitives.

Comparisons with single-level baselines. We also implemented Discriminator Actor-Critic (DAC) [110] with access to a single demonstration in the pick and place, push, hollow and kitchen environments, to analyse how PIPER fares against single-level baselines

trained with privileged information. Additionally, we compared PIPER with a single-level RL baseline implemented using SAC. Both baselines failed to show significant progress, while PIPER clearly outperforms the baselines. This clearly shows the advantage of hierarchical abstraction employed by PIPER.

7.2.3 Qualitative Insights

- **Mitigation of Reward Non-Stationarity through Preference-Based Relabeling:** Unlike baseline hierarchical RL methods that suffer from destabilization caused by changing lower-level policies, PIPER employs a learned preference-based reward model to relabel higher-level replay buffers. This decouples the higher-level reward signal from lower-level policy non-stationarity. Qualitatively, this leads to smoother and more stable policy improvement over training phases as the higher-level policy receives consistent feedback, enabling reliable subgoal selection throughout learning.
- **Primitive-Informed Regularization Ensuring Feasible Subgoals:** A key qualitative difference is PIPER’s primitive-informed regularization, which conditions the subgoal generator on the current capabilities of the lower-level policy via its value function. This approach prevents the generation of unrealistic or infeasible subgoals that halt learning at lower levels, which a common failure mode in other hierarchical approaches. As a result, the hierarchy’s decomposition of the problem maintains a meaningful balance between levels, ensuring progressive skill acquisition without degenerate task splits.
- **Hindsight Relabeling Enhancing Sparse Reward Learning:** PIPER’s use of hindsight relabeling in the preference feedback loop enriches sparse environments with denser, more informative signal. This strategy qualitatively speeds up the learning process by enabling the agent to reinterpret past experiences with alternate subgoals, resulting in more effective exploration and policy updates. Such behaviors are noticeably absent in other flat or vanilla hierarchical algorithms.

- **Hierarchical Preference-Based Learning Outperforms Flat Approaches:**

Unlike flat preference-based RL methods which directly optimize policies from preferences over primitive actions, PIPER’s hierarchical structure leverages subgoal abstractions to focus learning on temporally extended objectives. This decomposition allows more efficient credit assignment over long horizons and better exploration strategies, thus enabling improved performance.

- **Use of Soft Target Updates for Reward Model Stability:**

The inclusion of soft target networks to stabilize the preference-based reward model is a deliberate architectural refinement not typically found in prior preference-based learning approaches. This qualitative improvement prevents oscillatory reward signals and catastrophic forgetting in reward estimation.

7.2.4 Ablation Analysis

Here, we empirically demonstrate the importance of our each various component technique, by performing corresponding ablation study. We analyze the effect of varying the weight parameter α in Figure ??, and found that setting appropriate value of hyperparameter α is crucial for improved performance. If α is too small, we lose the advantages of primitive-informed regularization, leading to poor performance. In contrast, if α is too large, it leads to degenerate solutions. We also analyze the effect of removing the hindsight relabeling (PIPER-No-HR ablation). In Figure 7.4, the empirical results show that hindsight relabeling indeed leads to improved performance. Further, we remove the target networks from PIPER (PIPER-No-Target), to analyze whether using target networks stabilizes training. As seen in Figure 7.5, target networks significantly improve training stability. Finally, in Figure 7.6, we compare PIPER against two baselines with well-behaved, hand designed dense reward functions ((i) flat SAC with dense reward function (SAC-Dense), and (ii) PEBBLE with actual human preferences (PEBBLE-Dense)). This comparison aims to aid the practitioners for choosing between hand-designing dense reward functions, or implementing PIPER. Although SAC-Dense outperforms PIPER in easier tasks, PIPER is

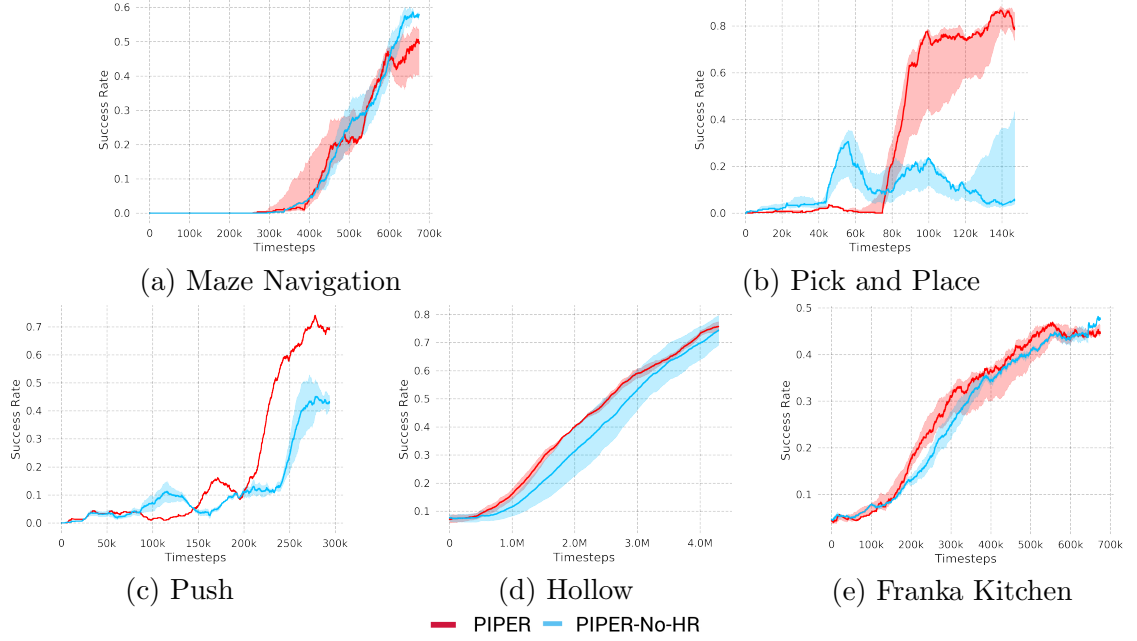


FIGURE 7.3: **Hindsight Relabeling ablation** This figure compares the performance of our PIPER approach with PIPER-No-HR ablation, which is effectively PIPER without hindsight relabeling [95] (as explained in Section 7.1.3) on four sparse maze navigation and robotic manipulation environments: (i) maze navigation (Row 1, Col 1), (ii) pick and place (Row 1, Col 2), (iii) push (Row 2, Col 1), (iv) hollow (Row 2, Col 2), and (v) franka kitchen (Row 2, Col 3). The plots showcase that although hindsight relabeling demonstrates minor performance improvement in sparse maze and kitchen tasks, it provides significant training speedup in sparse pick and place and push environments.

able to outperform SAC-Dense in harder environments. We believe this to be due to two reasons:

- hand-designing a suitable reward in complex environments is comparatively hard, and may lead to sub-optimal performance, and
- the advantages of various design choices in PIPER (exploration and task abstraction due to hierarchical structure, mitigating non-stationarity using preference based RL, reward densification using hindsight relabeling, feasible subgoal generation using primitive regularization, and added training stability due to target networks) outweigh the presence of dense rewards in SAC-Dense baseline.

Notably, SAC-Dense completely fails to solve the kitchen task. PEBBLE-Dense is unable to show good performance in any task. We also implemented PEBBLE with actual human

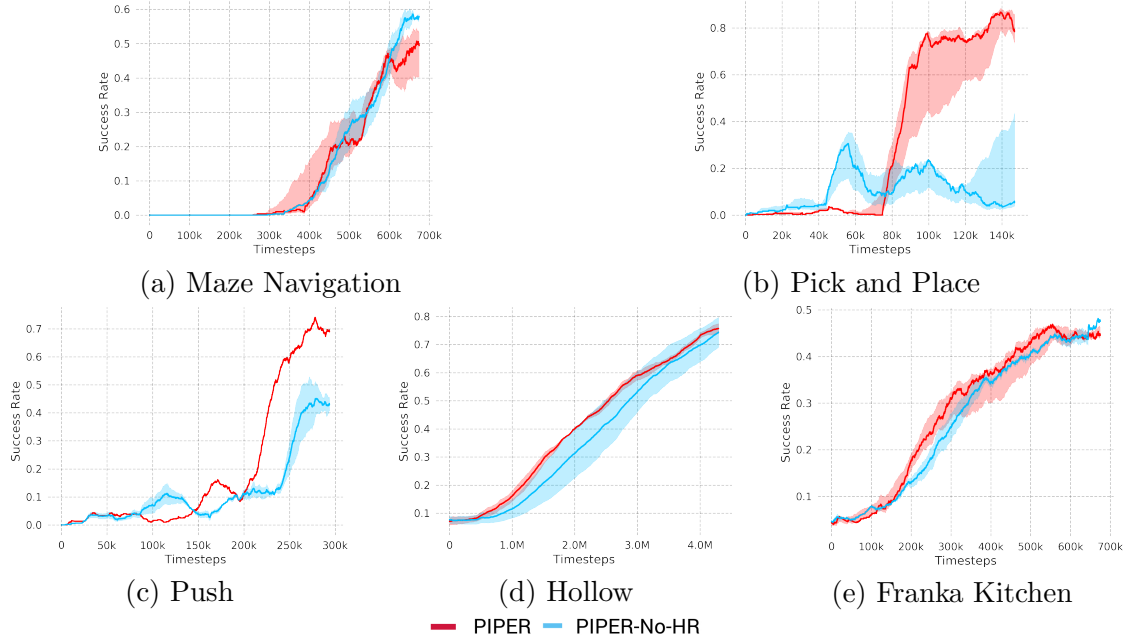


FIGURE 7.4: **Hindsight Relabeling ablation** This figure compares the performance of our PIPER approach with PIPER-No-HR ablation, which is effectively PIPER without hindsight relabeling [95] (as explained in Section 7.1.3) on four sparse maze navigation and robotic manipulation environments: (i) maze navigation (Row 1, Col 1), (ii) pick and place (Row 1, Col 2), (iii) push (Row 2, Col 1), (iv) hollow (Row 2, Col 2), and (v) franka kitchen (Row 2, Col 3). The plots showcase that although hindsight relabeling demonstrates minor performance improvement in sparse maze and kitchen tasks, it provides significant training speedup in sparse pick and place and push environments.

preferences (rather than primitives), but find that this baseline is unable to show good performance in any of the tasks. This shows that although PEBBLE shows impressive performance in simple environments, it needs further improvements to make it work in complex robotic manipulation tasks like the ones explored in this work.

7.3 Related Work

Preference-based Learning

Preference-based reinforcement learning (PbRL) has emerged as a compelling paradigm for learning from non-numeric, relative forms of human feedback, such as pairwise rankings or preferences, to address the challenges of reward engineering and task specification in RL. Unlike traditional RL, which depends on a manually designed reward function, PbRL

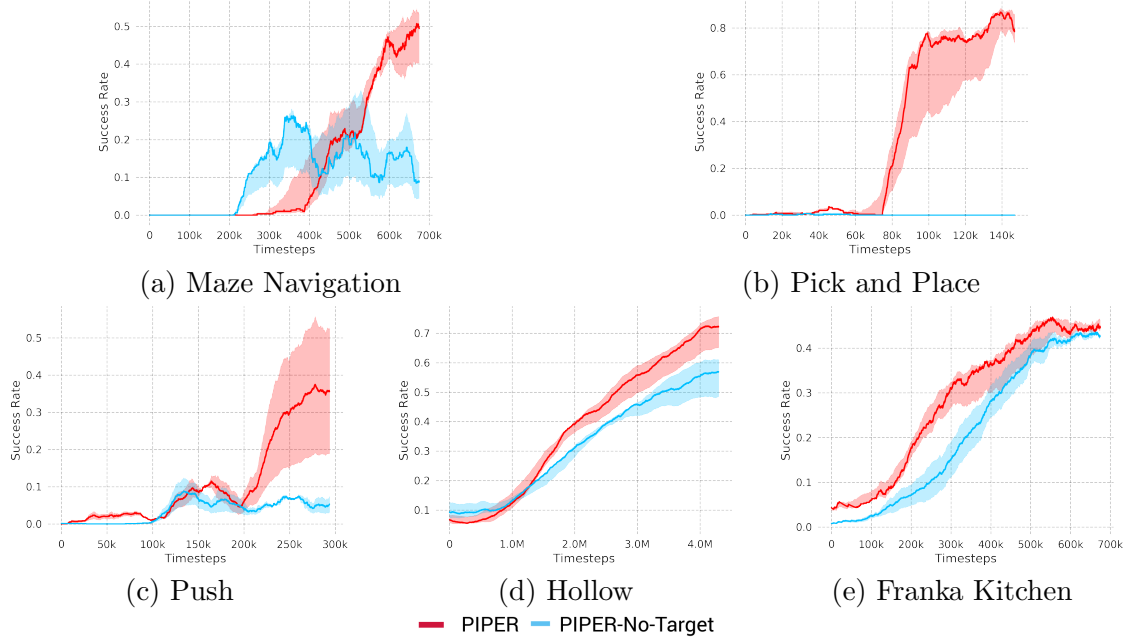


FIGURE 7.5: **Target networks ablation** This figure compares the performance of our PIPER approach with PIPER-No-Target ablation, which is effectively PIPER without target networks implementation [154] on four sparse maze navigation and robotic manipulation environments: (i) maze navigation (Row 1, Col 1), (ii) pick and place (Row 1, Col 2), (iii) push (Row 2, Col 1), (iv) hollow (Row 2, Col 2), and (v) franka kitchen (Row 2, Col 3). The plots showcase that using target networks significantly improves performance and indeed reduces training instability caused by non-stationary reward models $\hat{r}_\phi$, learnt using preference based learning.

enables agents to optimize behaviors directly based on human judgments, thereby bypassing the intricacies of reward shaping and making RL applicable to a broader spectrum of tasks and end-users [158, 159].

Direct Human Feedback and the TAMER Framework. Initial studies, notably the TAMER framework [160], demonstrated that RL agents can be trained interactively through scalar feedback signals from non-expert users. The TAMER+RL extension integrates such feedback with environmental rewards, improving agent performance and learning speed [161]. Extensions of TAMER further support continuous state and action spaces [162], showing the breadth of applicability.

Warnell *et al.* [163] extended TAMER to deep RL, enabling neural agents to interpret and utilize feedback signals as a surrogate reward, while Li *et al.* [164] and Knox *et al.* [161]

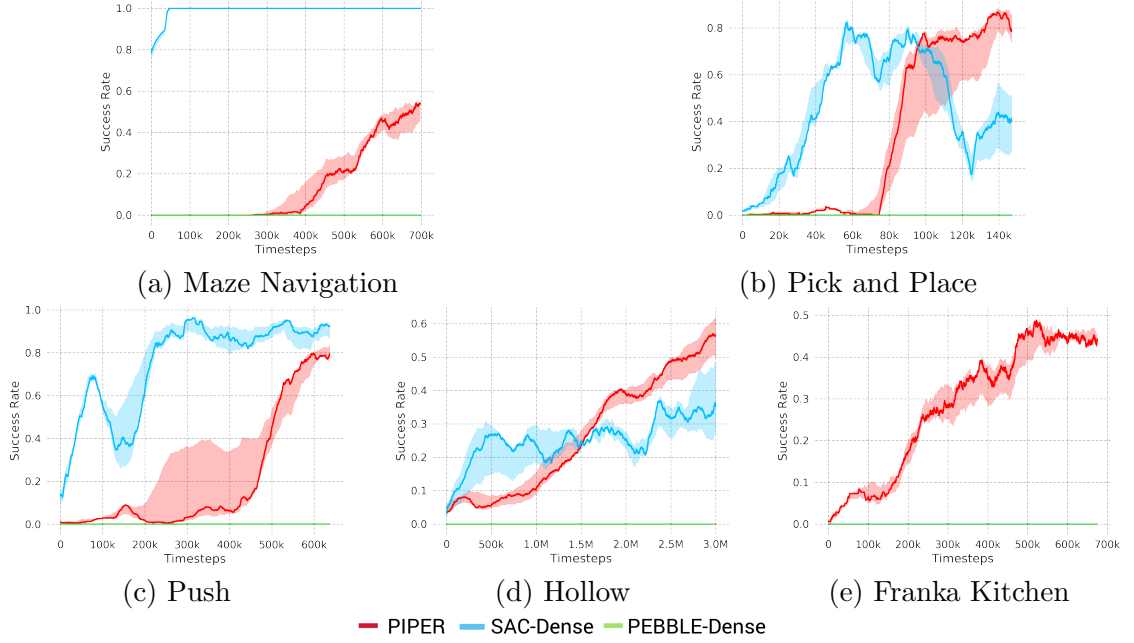


FIGURE 7.6: **Dense rewards ablation** This figure compares the success rate performances between PIPER and two baselines based on dense rewards: (i) SAC-Dense, and (ii) PEBBLE-Dense on four sparse maze navigation and robotic manipulation environments: (i) maze navigation (Row 1, Col 1), (ii) pick and place (Row 1, Col 2), (iii) push (Row 2, Col 1), (iv) hollow (Row 2, Col 2), and (v) franka kitchen (Row 2, Col 3). Although SAC-Dense outperforms PIPER on easier tasks, PIPER is able to outperform SAC-Dense in harder tasks. Notably, PEBBLE-Dense is unable to solve any of the tasks. Thus, PIPER shows impressive performance, and is a viable approach in complex sparse reward scenarios.

studied the impact of informative behavior and training interface design for boosting the efficiency and quality of human feedback.

Learning from Preference Rankings/Pairwise Comparisons. Several methods use human rankings for RL, with early work introducing probabilistic models of preference and incorporating them into RL policy updates [165–167]. One influential work [33] builds a reward model from pairwise preference comparisons, then uses it as a proxy reward in standard deep RL algorithms.

On-Policy and Off-Policy Approaches. On-policy methods, as in [33], rely on feedback collected in lock-step with policy exploration, often requiring considerable data. Off-policy PbRL and RLHF approaches [168–172] improve sample efficiency by decoupling

data collection from feedback elicitation, allowing for greater data re-use and better scaling to expensive or delayed feedback scenarios.

Methods like PEBBLE [168] and LEASE [170] leverage offline datasets and innovative preference augmentation for efficient offline training. Newer works provide theoretical analysis of sample complexity and policy evaluation guarantees when learning rewards from preferences rather than explicit signals [171, 172]. Additional improvements include efficient querying and preference aggregation [173, 174], as well as studies into the robustness and reliability of learning from diverse, potentially noisy or crowd-sourced preferences [174].

In this work, we build on these foundations by proposing an off-policy hierarchical RL approach using preference-based feedback to mitigate non-stationarity in HRL while regularizing subgoal generation through primitive-informed regularization.

7.4 Discussion

Limitations: PIPER uses L2 distance between states as the informative metric, which might be hard to compute in scenarios where the subgoals and goals are high-dimensional (e.g. images). A possible way to deal with this is to find a near optimal latent representation [175]. Further, although PIPER replaces human-in-the-loop preferences via primitive-in-the-loop (PiL) approach to generate trajectory preferences, human preferences may contain additional information that PiL may be unable to capture (say, humans may prefer trajectories that are *safe* to traverse, which do not involve any wall collisions). A possible way to deal with this issue is to employ additional rewards in Eqn 7.5, which prefer a safe trajectory. Similarly, the proposed primitive regularization approach may not generate such *safe* subgoals in its naive form. To this end, we can additionally learn a new *safe* value function $V^s(s, g)$, which regularizes the higher agents to produce *safe* subgoals. Finally, the proposed target networks require additional memory and training overhead, and require setting the hyper-parameter τ . However, we found in practice that

the memory and training overheads are minimal, and setting τ does not cause a significant overhead.

Conclusion: In this work, we propose PIPER, a primitive informed hierarchical RL algorithm that leverages preference based learning to mitigate the non-stationarity issue in HRL. Using primitive informed regularization, PIPER is able to generate efficient subgoals, according to the goal achieving capability of the lower primitive. We demonstrate that by incorporating primitive informed regularization, hindsight relabeling, and target networks, PIPER is able to perform complex robotic tasks in sparse scenarios, and significantly outperform the baselines. Additionally, PIPER is able to outperform flat preference based learning. We therefore believe that hierarchical preference based learning is a promising step towards building practical robotic systems that solve complex real-world tasks.

Chapter 8

Hierarchical Preference

Optimization: Learning to Achieve

Goals via Feasible Subgoals

Prediction

The strongest principle of growth lies in human choice.

– George Eliot

¹ In the last chapter, we propose an RLHF-based method to mitigate non-stationarity in HRL by first learning a reward model from preference data and then using RL to train the high-level policy. In contrast, direct preference optimization (DPO) methods [34] learn policies directly from preference data, bypassing the need for expensive and potentially unstable RL step which can still result in infeasible subgoal predictions for the lower-level primitives. Leveraging DPO to learn the high-level policy in HRL could efficiently mitigate non-stationarity. However, naively applying DPO requires effective pre-trained reference

¹This chapter is based on the paper in submission at NeuRIPS 2025 [80].

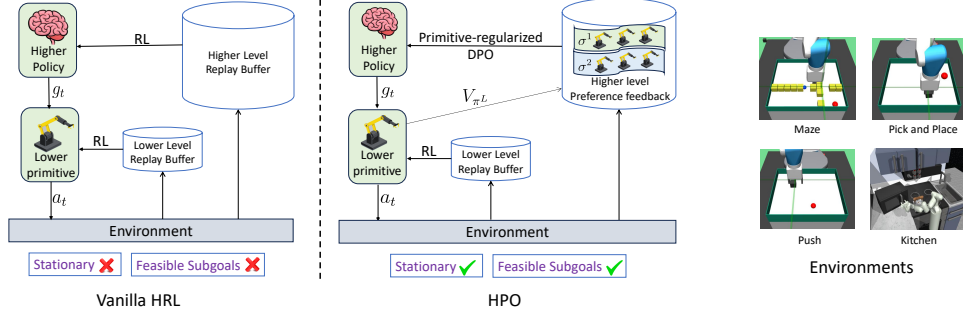


FIGURE 8.1: **HPO Overview:** **(left)** In vanilla HRL, the higher level rewards depend on the environment rewards, and thus on the lower primitive behavior, which causes non-stationarity in HRL. Also, the higher level may predict infeasible subgoals that are too hard for lower primitive. **(middle)** In HPO, the lower level value function V_{π_L} is leveraged to condition higher level policy into predicting feasible subgoals, and direct preference optimization (DPO) is used to optimize higher level policy. Since this preference-based learning approach does not depend on lower primitive, this mitigates non-stationarity. Note that since the *current* estimation of value function is used to regularize the higher policy, it does not cause non-stationarity. **(right)** Training environments: (i) maze navigation, (ii) pick and place, (iii) push, and (iv) franka kitchen environment.

policies, which are often unavailable in robotics scenarios. Additionally, such approaches may still produce subgoals that are infeasible for the lower-level policy.

In this chapter, we propose Hierarchical preference optimization (HPO), a hierarchical approach for addressing non-stationarity (C1) and infeasible subgoal generation (C2), that optimizes the higher level policy using token-level DPO objective [176], and the lower level primitive policy using RL. HPO addresses non-stationarity (C1) by leveraging preference data to learn the higher level policy, thereby avoiding dependence on changing lower level primitive. We derive token-level DPO objective in a maximum entropy RL setting, to eliminate the requirement of pre-trained models as reference policy. Furthermore, in order to address (C2) and generate feasible subgoals, we formulate HRL as a bi-level formulation, which regularizes the higher level policy to predict feasible subgoals using lower level value function based regularization, which we call *primitive regularization*. Finally, since eliciting human feedback is hard in complex robotic tasks, we employ primitive-in-the-loop approach [79] for autonomously generating preferences using sparse environment rewards. The primary contributions of this work are as follows.

- (i) **Novel primitive-regularized preference optimization approach to solve hierarchical RL:** We propose a novel Hierarchical Preference Optimization (HPO) method that leverages primitive-regularized Direct Preference Optimization (DPO) to solve complex RL tasks using human preference data (Section 8.1). Our approach is principled; we derive it by reformulating the HRL problem as a bi-level optimization problem. To the best of our knowledge, this is the first work to utilize the bi-level optimization framework to develop a principled solution for HRL.
- (ii) **Addressing non-stationarity and infeasible subgoal generation issues of SoTA HRL:** By adopting a principled approach, we solve the bi-level optimization problem while respecting the distributional dependence between lower-level and higher-level policies. This enables our approach to significantly reduce the effects of non-stationarity and infeasible subgoal generation in a variety of scenarios. We provide detailed evidence of this in Section 8.2 Figure 8.3.
- (iii) **Improved performance in complex robotics tasks:** Our extensive experiments demonstrate that HPO demonstrates an improvement of upto 40% over the baselines in most of the task environments, where other baselines typically fail to make any significant progress (Section 8.2).

Background

MDP Setup.

We formulate the problem using a Markov Decision Process (MDP), defined as $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$. Here, $\mathcal{S}$ represents the state space, and $\mathcal{A}$ denotes the action space. The transition dynamics are governed by the stochastic probability function $p : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$, which maps each state-action pair to a probability distribution over subsequent states. The reward function, $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, provides a scalar reward based on the current state and action, and $\gamma \in (0, 1)$ is the discount factor that modulates the importance of future rewards.

At each time step t , the agent follows a policy $\pi : \mathcal{S} \rightarrow \Delta(\mathcal{A})$, which defines a probability distribution over actions a_t given the current state s_t . After taking action a_t , the agent receives a reward $r_t = r(s_t, a_t)$, and the environment transitions to a new state s_{t+1} according to the transition probability $p(\cdot|s_t, a_t)$.

RL Objective.

The objective in RL is to find an optimal policy that maximizes the expected cumulative reward, formally defined as $\pi^* := \arg \max_{\pi} J(\pi)$, where $J(\pi) = \mathbb{E}_{\pi} [\sum_{t=0}^{\infty} \gamma^t r_t]$. The value function for policy π , denoted as $V_{\pi}(s_t, g_t)$, captures the expected cumulative reward from state s_t with goal g_t , and is defined as $V_{\pi}(s_t, g_t) = \mathbb{E}_{\pi} [\sum_{t=0}^T \gamma^t r_t | s_t, g_t]$, where the expectation is over policy trajectories, γ is the discount factor, and r_t is the reward at time t . The Q function for policy π is denoted as $Q_{\pi}(s_t, g_t, a_t)$, which is the expected reward over trajectories after taking action a_t at current time-step t .

Goal-Conditioned HRL.

The HRL framework defines two MDPs, a higher-level MDP and a low-level MDP, which we define as follows:

Higher Level MDP: The higher level MDP is defined as $(\mathcal{S}, \mathcal{G}, \mathcal{A}^H, p^H, r^H, \gamma^H)$, where $\mathcal{S}$ is the state space, and $\mathcal{G}$ is the goal space. The higher level action space is denoted as $\mathcal{A}^H$, where the higher-level policy $\pi^H : \mathcal{S} \rightarrow \Delta(\mathcal{A}^H)$ generates subgoals $g_t \sim \pi^H(\cdot|s_t)$ for the lower-level policy. In our case, we consider $\mathcal{A}^H = \mathcal{G}$. Further, the higher level transition dynamics are denoted as $p^H : \mathcal{S} \times \mathcal{G} \times \mathcal{A}^H \rightarrow \Delta(\mathcal{S})$. The higher-level policy is rewarded based on the environment reward $r_t^H(s_t, g^*, g_t)$, where $g^* \in \mathcal{G}$ is the final goal. The reward function for the higher level, $r^H : \mathcal{S} \times \mathcal{G} \times \mathcal{A}^H \rightarrow \mathbb{R}$, encourages progress toward the final goal objective. It is worth highlighting here that although the rewards for the higher-level policy are derived from the environment reward, due to the inherent hierarchical structure and since the next achieved state for the higher level policy π^H depends on the level policy

π^L , the rewards achieved by the higher level policy indirectly depend on the current lower-level policy π^L . We explicitly state this nuance by representing the higher level reward as $r_{\pi^L(\cdot|s_t, g_t)}^H$ in Section 8.1 later.

Lower Level MDP: In contrast, the lower level MDP is defined as $(\mathcal{S}, \mathcal{A}^L, p^L, r^L, \gamma^L)$, where $\mathcal{S}$ is the state space, $\mathcal{A}^L$ is the action space, and $p^L : \mathcal{S} \times \mathcal{A}^L \rightarrow \Delta(\mathcal{S})$ denotes the transition dynamics. We operate under a sparse reward setting, where the lower-level policy is driven by a sparse reward signal $r_t^L(s_t, g_t, a_t) = \mathbf{1}_{\{|s_t - g_t|^2 < \varepsilon\}}$, with $\mathbf{1}_C$ as an indicator function returning 1 if the condition C holds, indicating that the subgoal is reached. Notably, the reward function $r^L : \mathcal{S} \times \mathcal{G} \times \mathcal{A} \rightarrow \mathbb{R}$ encourages progress towards achieving the subgoal g_t provided by the higher level policy.

Hierarchical Setup: The higher-level policy generates subgoals for the lower-level policy after every k timesteps, which then takes primitive actions to achieve these subgoals within the k timesteps, thereby collectively working towards achieving the overall objective. Formally, the higher-level policy $\pi^H : \mathcal{S} \rightarrow \Delta(\mathcal{G})$ selects a subgoal $g_t \in \mathcal{G}$, where $\mathcal{G}$ is a subset of the state space $\mathcal{S}$, representing the set of goals. A subgoal $g_t \sim \pi^H(\cdot|s_t)$ is selected every k timesteps, such that $g_t = g_{k \cdot \lceil t/k \rceil}$ and remains fixed during this period. At each timestep t , the lower-level policy $\pi^L : \mathcal{S} \times \mathcal{G} \rightarrow \Delta(\mathcal{A})$ selects an action $a_t \sim \pi^L(\cdot|s_t, g_t)$ based on the current state s_t and the subgoal g_t , after which the environment transitions to the next state, $s_{t+1} \sim p(\cdot|s_t, a_t)$. Thus, the higher level Q^H and V^H functions are the expected rewards defined over higher level subgoal prediction trajectories defined in the higher level replay buffer as tuples $(s_t, g^*, g_t, r_t^H, s_{t+k})$ (updated every k timesteps), and the the lower level Q^L and V^L functions are the expected rewards defined over lower level primitive action prediction trajectories defined in the lower level replay buffer as tuples $(s_t, g_t, a_t, r_t^L, s_{t+1})$.

Next, we enlist the challenges of standard HRL methods.

Challenges of HRL

While HRL offers advantages over traditional RL, such as better sample efficiency through temporal abstraction and enhanced exploration [6, 26], it also faces significant challenges. We focus on two key issues:

- C1.** Training instability due to *non-stationarity* in off-policy HRL, and
- C2.** Suboptimal performance due to the generation of *infeasible subgoals* by the higher-level policy.

Off-policy HRL suffers from non-stationarity because the behavior of the lower-level policy changes over time [26, 27]. This makes previously collected higher-level transitions in the replay buffer outdated, reducing their effectiveness. Furthermore, the higher-level policy may suggest infeasible subgoals for the lower-level policy [28], hindering learning and decreasing overall performance. Thus, despite its theoretical advantages, HRL often underperforms in practice [26, 27].

Preference-based learning (PbL) methods such as RLHF [33, 149, 150] and DPO [34] leverage preference data to solve complex tasks. Prior work [79] leverages RLHF to mitigate non-stationarity in HRL using preferences. However, directly applying PbL techniques to HRL remains challenging. We now first discuss PbL approaches and then discuss their key limitations when directly applying to HRL.

PbL: Preference Based Learning

Reinforcement Learning from Human Feedback (RLHF): In this setting, the agent behavior is represented as a k -length trajectory denoted as τ , of state observations and actions: $\tau = ((s_t, a_t), (s_{t+1}, a_{t+1}) \dots (s_{t+k-1}, a_{t+k-1}))$. The reward model to be learned is denoted by $\hat{r}_\phi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, where ϕ are the reward model parameters. Thus, the

preferences between two trajectories, τ^1 and τ^2 , can be expressed through the Bradley-Terry model [153]:

$$P_\phi[\tau^1 \succ \tau^2] = \frac{\exp \sum_t \hat{r}_\phi(s_t^1, a_t^1)}{\sum_{i \in \{1,2\}} \exp \sum_t \hat{r}_\phi(s_t^i, a_t^i)}, \quad (8.1)$$

where $\tau^1 \succ \tau^2$ implies that τ^1 is preferred over τ^2 . The preference dataset $\mathcal{D}$ has entries of the form (τ^1, τ^2, y) , where $y = (1, 0)$ when τ^1 is preferred over τ^2 , $y = (0, 1)$ when τ^2 is preferred over τ^1 , and $y = (0.5, 0.5)$ in case of no preference. In RLHF, we first learn the reward function $\hat{r}_\phi$ [33] using cross-entropy loss along with use (8.1) to yield the formulation:

$$\mathcal{L}(\phi) = -\mathbb{E}_{(\tau^1, \tau^2, y) \sim \mathbb{D}} [\log \sigma(\sum_{t=0}^{T-1} \hat{r}_\phi(s_t^1, g^*, g_t^1) - \sum_{t=0}^{T-1} \hat{r}_\phi(s_t^2, g^*, g_t^2))]. \quad (8.2)$$

Subsequently, the reward model is used to learn the corresponding policy using RL.

Direct Preference Optimization (DP0): Although RLHF provides an elegant framework for learning policies from preferences, it involves RL training step which is often expensive and unstable in practice. In contrast, DP0 [34] circumvents the need for RL step by using a closed-form solution for the optimal policy of the KL-regularized RL problem [155]: $\pi^*(a|s) = \frac{1}{Z(s)} \pi_{ref}(a|s) e^{r(s,a)}$, where π_{ref} is the reference policy, π^* is the optimal policy, and $Z(s)$ is a normalizing partition function ensuring that π^* provides a valid probability distribution over $\mathcal{A}$ for each $s \in \mathcal{S}$. This formulation is rearranged to yield the reward function: $r(s, a) = \alpha \log \pi^*(a|s) - \alpha \log \pi_{ref}(a|s) - Z(s)$, which is then substituted in the standard cross-entropy loss (8.2), to yield the following objective:

$$\mathcal{L}_{DPO}(\theta) = -\mathbb{E}_{(s, y_1, y_2) \sim \mathbb{D}} [\log \sigma(\alpha \log \frac{\pi_\theta(y_1|s)}{\pi_{ref}(y_1|s)} - \alpha \log \frac{\pi_\theta(y_2|s)}{\pi_{ref}(y_2|s)})], \quad (8.3)$$

where θ are the policy parameters and σ denotes the sigmoid function. Next we discuss key limitations when directly applying PbL to HRL.

Limitations of directly applying PbL to HRL

Directly using RLHF: Prior approaches leverage the advancements in PbL to mitigate non-stationarity [79] by utilizing the reward model r_ϕ^H learned using RLHF as higher level rewards instead of environment rewards $r_{\pi_L}^H$ used in vanilla HRL approaches, which depend on the sub-optimal lower primitive. However, such approaches may lead to degenerate solutions by generating infeasible subgoals for the lower-level primitive. Additionally, such approaches require RL as an intermediate step, which might cause training instability [34].

Directly using DPO: In temporally extended task environments like robotics, directly extending DPO to the HRL framework is also non-trivial due to three reasons: (i) such scenarios deal with multi-step trajectories involving stochastic transitions models, (ii) efficient pre-trained reference policies are typically unavailable in robotics, (iii) similar to RLHF, such approaches may produce degenerate solutions when the higher level policy subgoal predictions are infeasible.

8.1 HPO : Hierarchical Preference Optimization

To deal with the challenges of HRL and utilizing PbL in HRL framework (8), we introduce **HPO : Hierarchical Preference Optimization**, a novel approach that leverages primitive regularized DPO to mitigate non-stationarity and infeasible subgoal generation issues in HRL. The rest of this section proceeds as follows: we first critically analyze the key source of non-stationarity in HRL and formulate HRL as a bi-level optimization problem. Utilizing this formulation, we then derive a primitive-regularized token-level DPO objective that overcomes the limitations of non-stationarity and infeasible subgoals generation in HRL. Finally, we analyze the HPO objective and gradient, and provide the final algorithm.

8.1.1 HRL: Bi-Level Formulation

We first consider the bi-level formulation for HRL. Let T be the task horizon, s_t be the state at time t , g^* be the final goal, and $g_t \sim \pi^H(\cdot|s_t, g^*)$ be the higher level subgoal prediction at time t . In the HRL framework, the lower level optimal policy is represented as π_*^L . The higher-level policy π^H predicts subgoal g_t to π_*^L , and receives higher level rewards $r_{\pi_*^L}^H$ (which depends on the optimal lower primitive π_*^L). The lower primitive policy executes primitive actions a_t to achieve the predicted subgoals. The higher level objective can thus be represented as:

$$\max_{\pi^H} \mathbb{E}_{\pi^H} \left[\sum_{t=0}^{T-1} r_{\pi_*^L(\cdot|s_t, g_t)}^H(s_t, g^*, g_t) \right]. \quad (8.4)$$

Issue of non-stationarity. In vanilla HRL, when the hierarchical levels are trained concurrently, since we do not have access to the optimal $\pi_*^L(\cdot|s, g)$ policy in the objective in (8.4), the higher level rewards are instead generated using an approximated lower primitive $\pi_\theta^L(\cdot|s, g)$, which is sub-optimal. Due to this sub-optimal lower level primitive, the higher level rewards $r_{\pi_\theta^L}^H$ are non-stationary (e.g. we may encounter different higher level rewards for the same subgoal prediction, as $\pi_\theta^L(\cdot|s, g)$ changes). This causes the non-stationarity issue in HRL.

Issue of infeasible subgoal generation. Since the sub-optimality in the lower-level policy affects its ability to reach a given subgoal, it consequently impacts the higher-level credit assignment during subsequent subgoal generation. This may cause the higher level to produce infeasible subgoals.

Distribution shift issue and bi-level formulation. To understand further the technical reasoning of the above-mentioned issues, let us expand upon the nested structure of the objective in (8.4), which is a bi-level optimization problem given by

$$\max_{\pi^H} \mathcal{J}(\pi^H, \pi_*^L(\pi^H)) \quad s.t. \quad \forall t \in [0 : T - 1], \pi_*^L(\cdot|s_t, g_t) = \operatorname{argmax}_{\pi^L(\cdot|s_t, g_t)} V_{\pi^L(\cdot|s_t, g_t)}(\pi^H), \quad (8.5)$$

where $\mathcal{J}(\pi^H, \pi_*^L(\pi^H))$ represents the higher level maximization objective (cf. (8.4)), and $V_{\pi^L}(\pi^H)$ is the lower level value function, conditioned on the higher level policy subgoals. We note that the problem in (8.5) is just an expanded version of the problem in (8.4). From the bi-level structure in (8.5), we note that when we approximate the optimal lower level policy π_*^L with an approximated version π_θ^L , it would actually break the nested structure of the problem in (8.5), leading to the issues of non-stationarity as well as infeasible subgoal generation. In this work, we propose a novel approach to solving the original HRL problem in (8.5) without breaking the nested structure as follows.

Directly considering the nested structure in (8.4). Utilizing the recent advancements in the optimization literature [177], we consider the equivalent constraint formulation of the problem in (8.5) as follows

$$\max_{\pi^H, \pi^L} \mathcal{J}(\pi^H, \pi^L) \quad s.t. \quad \forall t \in [0 : T - 1], \quad V_{\pi^L(\cdot|s_t, g_t)}(\pi^H) \geq V_{\pi_*^L(\cdot|s_t, g_t)}(\pi^H). \quad (8.6)$$

The constraint optimization problem is also challenging because it requires the access to $V_{\pi_*^L}(\pi^H)$ which is hard to obtain in practice. But let us try to discuss the constraint in detail. The constraint $V_{\pi^L}(\pi^H) \geq V_{\pi_*^L}(\pi^H)$ is essentially to enforce the optimality of the lower level policy, because $V_{\pi_*^L}(\pi^H)$ is the maximum (optimal) value. But we note that for the sparse reward scenarios (wlog assuming reward is between 0 and 1), the value function $V_{\pi^L}(\pi^H) = 0$ for any non-goal reaching policy. But $V_{\pi_*^L}(\pi^H) > 0$ or equivalently is definitely non-zero for the goal reaching optimal policy π_*^L . Therefore, it is sufficient to ensure $V_{\pi^L}(\pi^H) > 0$ in the constraint. To make the problem tractable, we introduce a parameter delta and reformulate the problem in (8.6) as follows:

$$\max_{\pi^H, \pi^L} \mathcal{J}(\pi^H, \pi^L) \quad s.t. \quad \forall t \in [0 : T - 1], V_{\pi^L(\cdot|s_t, g_t)}(\pi^H) \geq \delta, \quad (8.7)$$

where $\delta > 0$. Now, we formulate (8.7) as the following Lagrangian objective and substitute the objective from (8.4), where λ is the regularization weight hyper-parameter:

$$\max_{\pi^H, \pi^L} \mathbb{E}_{\pi^H} \left[\sum_{t=0}^{T-1} r_{\pi^L(\cdot|s_t, g_t)}^H(s_t, g^*, g_t) + \lambda_t (V_{\pi^L(\cdot|s_t, g_t)}(\pi^H) - \delta) \right]. \quad (8.8)$$

We can use (8.8) to solve the HRL policies for both higher and lower level, where (i) the higher level policy learns to achieve the final goal and predict feasible subgoals to the lower level policy, and (ii) the lower level policy learns to achieve the predicted subgoals. This will mitigate the issues of non-stationarity (**C1**) and infeasible subgoal generation (**C2**). However, directly optimizing (8.8) requires knowledge of higher level reward function $r_{\pi^L}^H$ as a function of π^L , which is unavailable. Also as mentioned before, using RL to optimize the objective will be unstable.

Next, we discuss HPO: our PbL based approach that efficiently optimizes (8.8) by learning a primitive-regularized DPO objective which simultaneously solves the issues of non-stationarity (**C1**) and infeasible subgoal generation (**C2**).

8.1.2 HPO

Here, we derive our HPO objective. When optimizing for the higher level policy, Equation (8.8) can be re-written as:

$$\max_{\pi^H} \mathbb{E}_{\pi^H} \left[\sum_{t=0}^{T-1} \hat{r}_\phi \right], \quad (8.9)$$

where, $\hat{r}_\phi = r_L^H(s_t, g^*, g_t) + \lambda(V_{\pi^L}(s_t, g_t) - \delta)$ (for simplicity, we will henceforth represent $r_{\pi^L(s_t, g_t)}^H$ by r_L^H). As originally explored in [178], the relationship between the future returns and the current timestep return can be captured by the following bellman equation corresponding to the optimal higher level policy π_*^H :

$$Q_*^H(s_t, g^*, g_t) = \begin{cases} r_L^H(s_t, g^*, g_t) + V_*^H(s_{t+1}, g^*) & \text{if } s_{t+1} \text{ is not terminal,} \\ r_L^H(s_t, g^*, g_t) & \text{if } s_{t+1} \text{ is terminal.} \end{cases} \quad (8.10)$$

Since there is a bijection between the reward function $r_L^H(s_t, g^*, g_t)$ and corresponding optimal state value function $Q_*^H(s_t, g^*, g_t)$ in the token MDP [176], we reformulate (8.10) to represent the rewards as follows:

$$r_L^H(s_t, g^*, g_t) = [Q_*^H(s_t, g^*, g_t) - V_*^H(s_{t+1}, g^*)]. \quad (8.11)$$

For the full trajectory τ , we can use (8.9) and (8.11) to get:

$$\begin{aligned} \sum_{t=0}^{T-1} \hat{r}_\phi &= \sum_{t=0}^{T-1} (Q_*^H(s_t, g^*, g_t) - V_*^H(s_{t+1}, g^*) + \lambda(V_{\pi^L}(s_t, g_t) - \delta)) \\ &\stackrel{(a)}{=} V_*^H(s_0, g^*) + \sum_{t=0}^{T-1} (Q_*^H(s_t, g^*, g_t) - V_*^H(s_t, g^*) + \lambda(V_{\pi^L}(s_t, g_t) - \delta)) \\ &\stackrel{(b)}{=} V_*^H(s_0, g^*) + \sum_{t=0}^{T-1} (A_*^H(s_t, g^*, g_t) + \lambda(V_{\pi^L}(s_t, g_t) - \delta)) \\ &\stackrel{(c)}{=} V_*^H(s_0, g^*) + \sum_{t=0}^{T-1} (\beta \log \pi_*^H(g_t | s_t, g^*) + \lambda(V_{\pi^L}(s_t, g_t) - \delta)), \end{aligned}$$

where (a) is due to adding and subtracting $V_*^H(s_0, g^*)$, (b) is due to $Q_*^H(s_t, g^*, g_t) - V_*^H(s_t, g^*) = A_*^H(s_t, g^*, g_t)$ where $A_*^H(s_t, g^*, g_t)$ represents the advantage function, and (c) is due to a result from [179], $A_*^H(s_t, g^*, g_t) = \beta \log(\pi_*^H(g_t | s_t, g^*))$. Substituting (8.12)

in (8.2) yields the primitive regularized DPO objective $\mathcal{L}_{\mathcal{O}} = \mathcal{L}(\pi_*^H, \mathbb{D})$ as:

$$\mathcal{L}_{\mathcal{O}} = -\mathbb{E}_{(\tau^1, \tau^2, y) \sim \mathbb{D}} \left[\log \sigma \left(\sum_{t=0}^{T-1} (\beta \log \pi_*^H(g_t^1 | s_t^1, g^*) - \beta \log \pi_*^H(g_t^2 | s_t^2, g^*) + \lambda V_{\pi^L}(s_t^1, g_t^1) - \lambda V_{\pi^L}(s_t^2, g_t^2)) \right) \right]$$

Note that terms $V_*^H(s_0, g^*)$ and $\lambda \delta$ are same for both trajectories and hence they cancel. This is the final HPO objective which optimizes the higher-level policy using primitive regularized DPO. Next, we analyze and discuss the HPO gradient.

Analyzing HPO gradient: We further analyze the HPO objective by interpreting the gradient $\mathcal{L}_{\mathcal{G}} = \nabla \mathcal{L}(\pi_*^H, \mathbb{D})$ with respect to higher level policy π_*^H , denoted as:

$$\begin{aligned} \mathcal{L}_{\mathcal{G}} = & -\beta \mathbb{E}_{(\tau_1, \tau_2, y) \sim \mathbb{D}} \left[\sum_{t=0}^{T-1} \left(\underbrace{\sigma(\hat{r}(s_t^2, g_t^2) - \hat{r}(s_t^1, g_t^1))}_{\text{higher weight for incorrect preference}} \right. \right. \\ & \cdot \left(\underbrace{\nabla \log \pi^H(g_t^1 | s_t^1, g^*)}_{\text{increase likelihood of } \tau_1} - \underbrace{\nabla \log \pi^H(g_t^2 | s_t^2, g^*)}_{\text{decrease likelihood of } \tau_2} \right) \left. \right) \Big]. \end{aligned} \quad (8.12)$$

where $\hat{r}(s_t, g_t, g^*) = \beta \log \pi^H(g_t | s_t, g^*) + \lambda V_{\pi^L}(s_t, g_t)$, which is the implicit reward determined by the higher-level policy and the lower-level value function. Conceptually, this objective increases the likelihood of preferred trajectories while decreasing the likelihood of dispreferred trajectories. Further, according to the strength of the KL constraint, the examples are weighted based on how inaccurately the implicit reward model $\hat{r}(s_t, g_t, g^*)$ ranks the trajectories. Notably, the λ weighted value function term V_{π^L} in the implicit reward $\hat{r}$ assigns high value to feasible subgoals, thus enabling primitive regularization.

Replacing human feedback: In the vanilla PbL framework, preferences are elicited from human feedback [33]. However, collecting large amount of human preference feedback required for PbL is computationally expensive. In this work, we follow the primitive-in-the-loop (PiL) approach in PIPER [79], and simulate this preference feedback by using implicit sparse rewards: $r^s(s_t, g^*, g_t)$, to determine preferences y between trajectories τ^1 and τ^2 . This feedback is obtained as follows: suppose the higher level policy π^H predicts

subgoal $g_t \sim \pi^H(\cdot|s_t, g^*)$ for state s_t and final goal g^* . The lower-level primitive executes primitive actions according to its policy π^L for k timesteps to end up in state s_{t+k-1} . We use the sparse reward provided by the environment at state s_{t+k-1} as the implicit sparse reward, i.e., $r^s(s_t, g^*, g_t) = \mathbf{1}_{\{\|s_{t+k-1} - g^*\|_2 \leq \epsilon\}}$. We use the sum of all k -step rewards r^s for comparing the two trajectories, to obtain preferences between higher level behavior sequences, as explained in detail in [79].

We now provide the psuedo-code for HPO algorithm in the Appendix 8.1.3 Algorithm 6.

8.1.3 HPO Algorithm

Here, we provide the pseudo-code for HPO algorithm

Algorithm 6 HPO	
1:	Initialize preference dataset $\mathcal{D} = \{\}$.
2:	Initialize lower level replay buffer $\mathcal{R}^L = \{\}$.
3:	for $i = 1 \dots N$ do
4:	// Collect higher level trajectories τ using π^H and lower level trajectories ρ using π^L ,
5:	// and store the trajectories in $\mathcal{D}$ and $\mathcal{R}^L$ respectively.
6:	// After every m timesteps, relabel $\mathcal{D}$ using primitive-in-the-loop feedback y .
7:	// Lower level value function update
8:	Optimize lower level value function V_{π_L} to get V_{π_L} .
9:	// Higher level policy update using HPO
10:	for each gradient step do
11:	// Sample higher level behavior trajectories.
12:	$(\tau^1, \tau^2, y) \sim \mathcal{D}$
13:	Optimize higher level policy π^U using (8.12).
14:	// Lower primitive policy update using RL
15:	for each gradient step do
16:	Sample ρ from $\mathcal{R}^L$.
17:	Optimize lower policy π^L using SAC.

8.2 Experiments

In our empirical analysis, we ask the following questions:

(1) **How well does HPO perform against baselines?** How well does HPO perform in

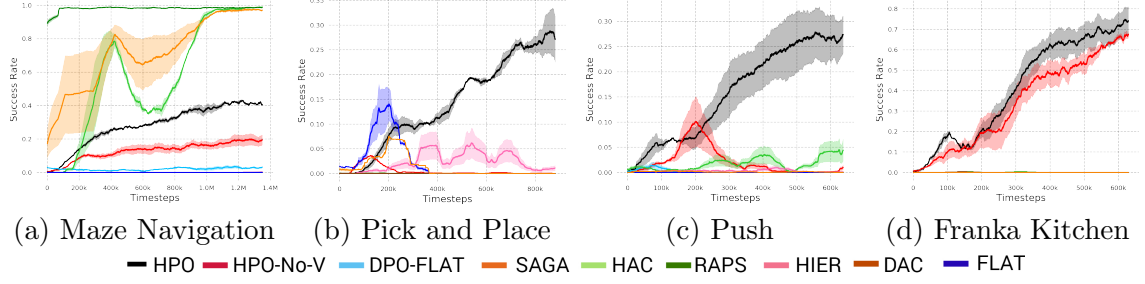


FIGURE 8.2: **Success Rate plots** This figure illustrates the success rates across four sparse-reward maze navigation and robotic manipulation tasks: maze navigation (Col 1), pick and place (Col 2), push (Col 3), franka kitchen (Col 4), where the solid lines represent the mean, and the shaded areas denote the standard deviation across 5 different seeds. We evaluate HPO against several baselines. Although HAC and RAPS surpass HPO in the maze task, HPO demonstrates strong performance and significantly outperforms the baselines in the more challenging tasks.

complex robotics control tasks against prior hierarchical and non-hierarchical baselines?

(2) Does HPO mitigate HRL limitations? How well does HPO mitigate the issues of non-stationarity (C1) and infeasible subgoal generation (C2) in HRL?

(3) What are the qualitative insights into HPO’s improved performance??

(4) What is the impact of our design decisions on the overall performance?

Can we concretely justify our design choices through extensive ablation analysis?

Task details: We assess HPO on four robotic navigation and manipulation environments: (i) maze navigation, (ii) pick and place [95], (iii) push, and (iv) franka kitchen environment [89]. These are formulated as sparse reward scenarios, where the agent is only rewarded when it comes within a δ distance of the goal. Due to this, these environments are hard where the agent must extensively explore the environment before coming across any rewards. As an example: in franka kitchen task, the agent only receives a sparse reward after achieving the final goal (e.g. successfully opening the microwave and then turning on the gas knob).

Environment details: The main objective of our empirical analysis is to evaluate our approach on complex sparsely rewarded long-horizon tasks. Hence, we re-formulate the maze navigation task and increase its complexity by considering randomly generated mazes. Thus, the agent has to learn to generalize across new mazes. In the franka kitchen task, the agent is sparsely rewarded only when it completes the final task, e.g open the microwave

and turn on the gas knob. These nuances prohibit the prior baselines from performing well in these tasks, which makes these test beds ideal scenarios for empirical evaluations. Unless otherwise stated, we maintain empirical consistency across all baselines to ensure fair comparisons. Finally, for harder tasks such as pick and place, push and franka kitchen, we assume access to one human demonstration and incorporate an imitation learning objective at the lower level to accelerate learning. However, we apply the same assumption consistently across all baselines to ensure fairness.

(1) How well does HPO perform against baselines?

In this section, we compare HPO against multiple hierarchical and non-hierarchical baselines. Please refer to Figure 8.2 for success rate comparison plots and subsequent discussion. The solid line and shaded regions represent the mean and standard deviation, averaged over 5 seeds.

Comparison with DPO Baselines.

HPO-No-V baseline: HPO-No-V is an ablation of HPO without primitive regularization, which uses DPO at the higher level to predict subgoals, and RL at the lower level policy. The primitive regularization approach in HPO regularizes the higher level policy to predict feasible subgoals. We employ this baseline to highlight the critical role of generating feasible subgoals. As shown in Figure 8.2, HPO outperforms this baseline, underscoring the critical role of feasible subgoal generation in achieving superior performance.

DPO-FLAT baseline: Here, we compare HPO with DPO-FLAT, which is a token-level DPO [176] implementation. Note that since we do not have access to a pre-trained model as a reference policy in robotics scenarios like generative language modeling, we use a uniform policy as a reference policy, which effectively translates to an additional objective of maximizing the entropy of the learnt policy. HPO is an hierarchical approach which benefits from temporal abstraction and improved exploration, as seen in Figure 8.2 which shows that HPO significantly outperforms this baseline.

Comparison with Hierarchical Baselines.

SAGA baseline: We compare HPO with SAGA [180], which is a hierarchical approach that employs state conditioned discriminator network training to address non-stationarity, by ensuring that the high-level policy generates subgoals that align with the current state of the low-level policy. We find that although SAGA performs well in the maze environment, it fails to solve harder tasks where HPO is able to significantly outperform it. This demonstrates that SAGA suffers from non-stationarity in harder long horizon tasks, and HPO is able to better mitigate non-stationarity issue in such tasks.

PIPER baseline: We compare HPO with PIPER [79], that leverages RLHF to learn higher level reward function to address HRL non-stationarity. Since HER [95] is an orthogonal approach to HPO, to ensure fair comparison, we implement an ablation of PIPER without HER. We find that HPO is able to outperform this PIPER ablation on all tasks. This demonstrates that our DPO based approach mitigates training instability caused by RL, and is able to better mitigate non-stationarity in HRL.

RAPS baseline: Here, we consider RAPS [133] baseline, which employs behavior priors at the lower level for solving the task. Although RAPS is an elegant framework for solving robotic tasks where behavior priors are readily available, it requires considerable effort to construct such priors and struggles to perform well in their absence, especially when dealing with sparse reward scenarios. Indeed we empirically find this to be the case, since although RAPS performs exceptionally well in maze navigation task, it fails to perform well in other sparse complex manipulation tasks.

HAC baseline: In order to analyze how well HPO addresses non-stationarity, we consider HAC [27] baseline. HAC tries to mitigate non-stationarity by simulating optimal lower level primitive behavior. This approach performs relabeling on the replay buffer transitions and is closely related to hindsight experience replay (HER) [95]. Although HAC performs well in maze task, it struggles to perform well in harder tasks. HPO outperforms this baseline in 3 out of 4 tasks.

HIER baseline: We also implement HIER, a vanilla HRL baseline implemented using SAC [114] at both hierarchical levels to demonstrate the significance of dealing with non-stationarity. As expected, HPO is able to consistently outperform this baseline in all tasks.

Comparison with Non-Hierarchical Baselines.

DAC baseline: We implement a single-level baseline: Discriminator Actor Critic (DAC) [110].

We provide one demonstration to DAC baseline in each environment. However, despite having access to privileged information, DAC still struggles to perform well.

FLAT baseline: We implement this baseline as a single-level SAC policy. This baseline fails to any progress, showing that our hierarchical structure is key to effective performance in complex tasks.

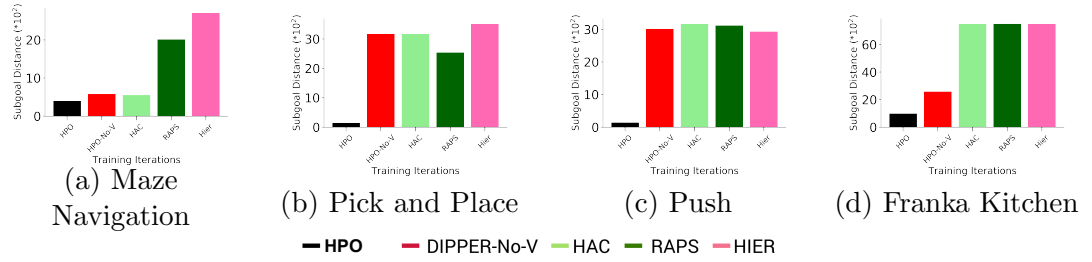


Figure 8.3: **Subgoal Distance Metric.** This figure compares HPO with HPO-No-V, HAC, RAPS, HIER baselines, based on average distance between subgoals predicted by the higher level policy and subgoals achieved by the lower level primitive. The tasks are (i) Maze Navigation (a), (ii) Pick and Place (b), (iii) Push (c) and (iv) Franka Kitchen environment (d). HPO consistently generates low average distance values, which implies that in HPO, the higher level policy generates achievable subgoals that induce optimal lower primitive goal reaching behavior. This shows that HPO is able to address non-stationary in HRL and generate feasible subgoals.

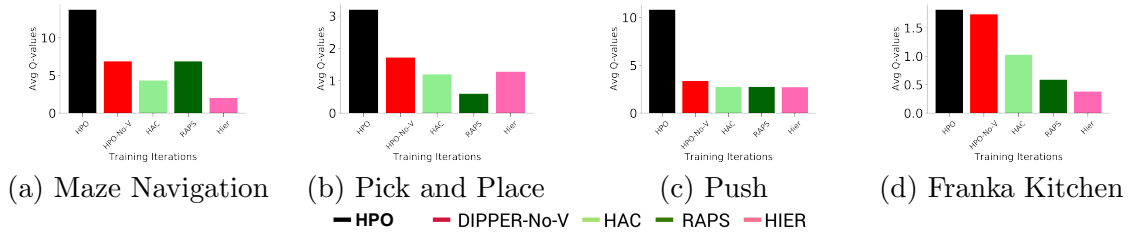


Figure 8.4: **Lower Q-Function Metric.** This figure compares HPO with HPO-No-V, HAC, RAPS, HIER baselines, based on average lower level Q function values for the subgoals predicted by the higher level policy. The tasks are (i) Maze Navigation (Col 1), (iii) Pick and Place (Col 2), (iii) Push (Col 3) and (iv) Franka Kitchen environment (Col 4). HPO consistently leads to large Q-function values, thus inducing optimal lower policy behavior and predicting feasible subgoals. Thus, HPO is able to mitigate non-stationary in HRL, while generating feasible subgoals.

(2) Does HPO mitigate HRL limitations?

Prior work largely lacks principled metrics for quantifying non-stationarity (C1) and infeasible subgoal generation (C2) in HRL. To address this gap, we introduce two novel

metrics specifically designed to measure these challenges. Using these metrics, we empirically demonstrate that HPO effectively mitigates both non-stationarity ((C1)) and infeasible subgoal generation ((C2)) in HRL.

Subgoal Distance Metric. We compare HPO with HPO-No-V, HAC, RAPS, HIER baselines on subgoal distance metric: the average distance between subgoals predicted by the higher level policy and subgoals achieved by the lower level primitive. A low average distance value implies that the predicted subgoals are feasible, thus inducing optimal lower level policy behavior (note that the optimal, lower-level policy is stationary, and thus inherently avoids non-stationarity issue). As seen in Figure 8.3, HPO consistently generates low average distance values, thus mitigating non-stationarity ((C1)). Further, low average distance values imply that in HPO, the higher level policy generates achievable subgoals for the lower primitive due to primitive regularization, thus addressing ((C2)).

Lower Q-Function Metric. We compare HPO with HPO-No-V, HAC, RAPS, HIER baselines on average lower level Q function values for the subgoals predicted by the higher level policy. High Q-values imply that the lower-level policy expects high returns for the predicted subgoals. Such subgoals are feasible and induce optimal lower primitive goal reaching behavior. As seen in Figure 8.4, HPO consistently leads to large Q-function values, which validates that HPO produces high-reward inducing and feasible subgoals, directly addressing both ((C1)) and ((C2)).

(3) What are the qualitative insights into HPO’s improved performance?

Beyond quantitative learning curves, there are several qualitative aspects in which HPO distinctly outperforms existing deep RL baselines by explicitly leveraging hierarchical structure and subgoal feasibility:

1. Principled Bi-level Optimization HPO frames hierarchical reinforcement learning as a rigorous bi-level optimization problem. Unlike many prior approaches that heuristically coordinate higher and lower-level policies, our formulation explicitly models their interdependent objectives. This ensures that subgoal selection by the higher-level policy is optimally tuned to the capabilities of the lower-level primitive, fostering principled

coordination across hierarchy levels.

2. Feasible Subgoal Generation via Primitive Regularization A key challenge in HRL is avoidance of infeasible subgoals that lower-level policies cannot achieve, which degrade performance and stability. HPO incorporates a primitive-regularization mechanism grounded in the lower-level value function. Unlike prior hierarchical baselines, this regularization constrains the higher-level policy to generate only subgoals that induce achievable behaviors in the lower-level policy, preventing degenerate or overly ambitious subgoal proposals.

3. Mitigation of Non-Stationarity through Direct Preference Optimization

Traditional HRL approaches suffer from non-stationarity because the higher-level rewards depend on evolving lower-level policies. To address this, HPO leverages direct preference optimization (DPO) to train the higher-level policy using preference feedback, effectively decoupling it from the non-stationary lower-level reward signals. This stabilizes training at the higher level and enables consistent policy improvement despite changes in the lower-level policy.

These qualitative insights emphasize HPO’s novel contributions to hierarchical reinforcement learning, highlighting how our method’s explicit modeling of hierarchy and subgoal feasibility facilitates superior performance and training stability in challenging real-world robotics tasks.

(4) What is the impact of our design choices?

We perform ablations to analyze our design choices. We first analyze the effect of varying regularization weight λ hyper-parameter in Figure 8.5. λ controls the strength of primitive regularization: if λ is too small, we lose the benefits of primitive regularization leading to infeasible subgoals prediction. Conversely, if λ is too large, the higher-level policy might fail to achieve the final goal by repeatedly predicting trivial subgoals. We also analyze the effect of varying β hyper-parameter in Figure 8.6. Excessive β causes over-exploration, preventing optimal subgoal prediction; whereas insufficient β limits exploration, risking suboptimal predictions.

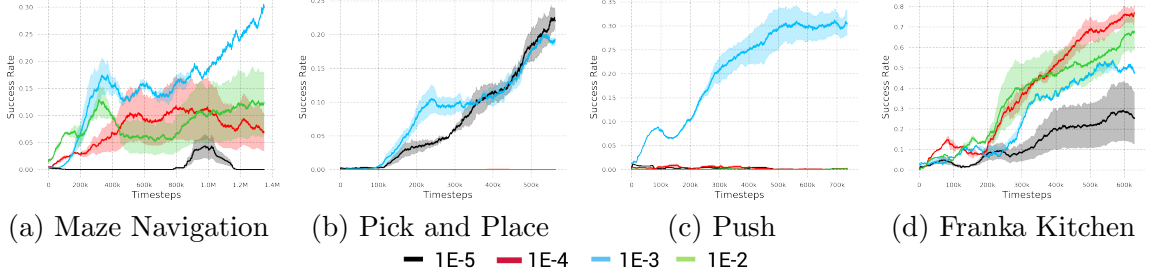


Figure 8.5: **Regularization weight ablation** This figure depicts the success rate performance for varying values of the primitive regularization weight λ on four environments: maze navigation (Col 1), pick and place (Col 2), push (Col 3), franka kitchen (Col 4). When α is too small, we lose the benefits of primitive-informed regularization resulting in poor performance, whereas too large α values can lead to degenerate solutions. Hence, selecting appropriate λ value is essential for accurate subgoal prediction and enhancing overall performance.

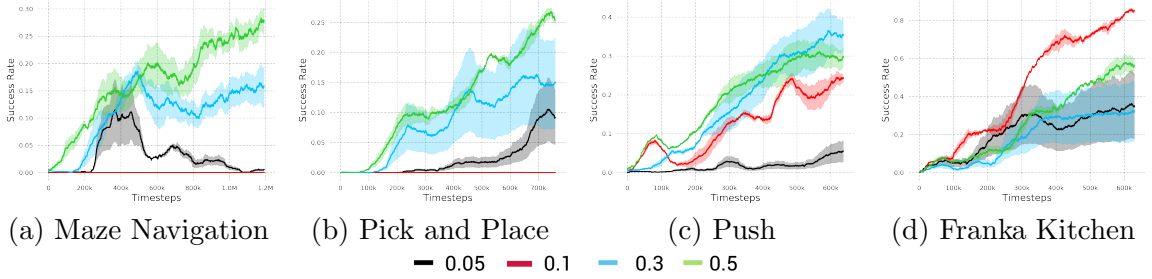


Figure 8.6: **Max-ent parameter ablation** This figure illustrates the success rate performance for different values of the max-ent parameter β hyper-parameter on four environments: maze navigation (Col 1), pick and place (Col 2), push (Col 3), franka kitchen (Col 4). This parameter controls the exploration in maximum-entropy formulation. If β is too large, the higher-level policy may perform extensive exploration but stay away from optimal subgoal prediction, whereas if β is too small, the higher-level might not explore and predict sub-optimal subgoals. Hence, selecting appropriate β value is essential for enhancing overall performance.

8.3 Related Work

Preference-based learning has emerged as a powerful avenue for applying reinforcement learning (RL) to domains where explicit reward engineering is difficult or infeasible. Instead of requiring numerical rewards, these approaches utilize human preference data, often in the form of pairwise comparisons or rankings—provided by annotators to guide agent learning.

A major breakthrough came with the demonstration that deep RL agents can learn from trajectory-level preference data, such as in the work of Christiano et al. [33]. Here, a

reward model is first trained from human preferences and then used as a surrogate for the true reward in subsequent RL training. This framework showed that complex behaviors could be learned in simulated and real-world environments solely from preference feedback.

As the field matured, researchers sought to improve the sample efficiency of preference learning. Notably, off-policy policy gradient and RLHF (Reinforcement Learning from Human Feedback) methods were proposed to better leverage previously collected data [114, 150]. These approaches decouple data collection from policy learning, making them more scalable to scenarios where human feedback is expensive.

A significant evolution in preference-based RL came with the introduction of Direct Preference Optimization (DPO) [34]. Unlike traditional RLHF, which requires a two-step process of reward model training followed by RL, DPO offers a direct approach: it optimizes the policy to align with human preferences using a KL-regularized maximum likelihood objective, circumventing the explicit reward modeling step. This formulation enables supervised policy training from preference data and simplifies training pipelines for large models.

Contrastive methods for preference optimization [181] have further advanced this line of work by employing contrastive losses and probabilistic modeling to efficiently use human feedback in policy optimization.

In this work, we build on the foundational knowledge in maximum entropy RL [179], and derive a token-level direct preference optimization [34, 176] objective regularized by lower-level primitive, resulting in an efficient hierarchical framework capable of solving complex robotic tasks.

8.4 Discussion

In this work, we introduce Hierarchical Preference Optimization HPO, a novel hierarchical approach that employs primitive-regularized direct preference optimization DPO to mitigate the issues of non-stationarity and infeasible generation in HRL. HPO employs token-level DPO to efficiently learn higher level policy using preference data, and RL to learn the lower

level primitive policy. Since the DPO formulation avoids the dependence on changing lower level primitive, it mitigates non-stationarity. We formulate HRL as a bi-level optimization objective to insure that the higher level policy generates feasible subgoals, and avoids degenerate solutions. Based on strong empirical results, We believe that HPO is an important step towards learning effective control policies for solving complex sparse robotics scenarios.

While HPO demonstrates effective non-stationarity mitigation using human preferences as a scalable supervision source beyond expert demonstrations (as in PEAR/CRISP), preferences still require pairwise judgments, limiting accessibility for non-expert users. This motivates extending hierarchical preference optimization to more intuitive, free-form guidance like natural language instructions, where Large Language Models (LLMs) can translate commands into stationary reward signals. In the next chapter, we introduce LGR2, which builds on HPO’s bi-level framework and primitive regularization by leveraging LLM-based Language-to-Reward (L2R) pipelines to generate semantically aligned rewards, further reducing supervision costs while preserving HRL benefits for complex, long-horizon robotics tasks.

Part IV

From Words to Actions

Chapter 9

LGR2: Language Guided Reward Relabeling for Accelerating Hierarchical Reinforcement Learning

Words may show a person's wit, but actions his meaning.

– Benjamin Franklin

Robotic systems capable of understanding and executing natural language instructions hold great promise for enabling intuitive human-robot interaction and flexible automation. Recent advances in deep reinforcement learning (RL) have demonstrated remarkable success in learning complex robotic behaviors from raw sensory inputs [91, 136, 182]. However, these methods often struggle with long-horizon tasks that require extensive planning and precise multi-step coordination, particularly under sparse reward signals where meaningful feedback is rare and delayed.¹

¹This chapter is based on the paper in submission at AAAI 2026 [81].

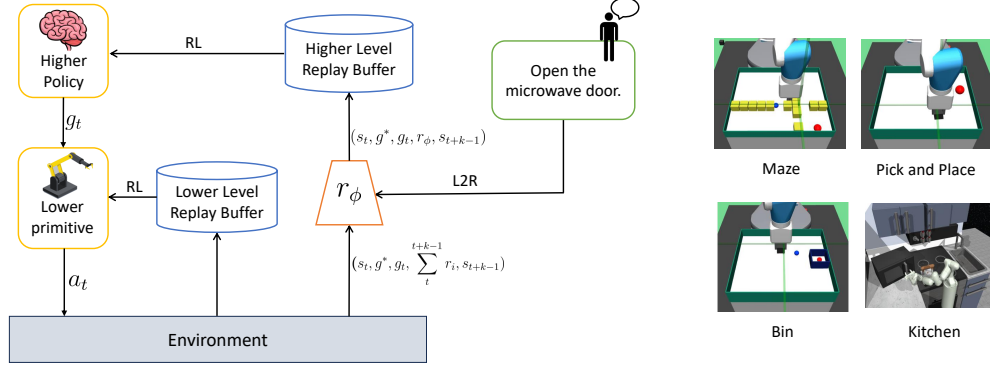


FIGURE 9.1: **LGR2 overview (left):** The higher-level policy predicts subgoals g_t for the lower-level policy, which executes primitive actions a_t on the environment. The lower-level replay buffer is populated by environment interactions, and lower-level policy is optimized by RL. L2R is used to translate human instructions to reward function parameters, which subsequently relabel the higher-level replay buffer transitions. Finally, RL is used to optimize the higher-level policy.

Environments (right): maze navigation, pick and place, bin, and franka kitchen.

Hierarchical reinforcement learning (HRL) provides a principled framework to tackle these challenges by decomposing tasks into temporally extended subgoals and learning nested policies [12, 65, 96]. This temporal abstraction improves exploration and credit assignment, essential for solving complex robotic control problems. Yet, conventional HRL algorithms face a significant hurdle: *non-stationarity*. As the lower-level policy evolves, the higher-level policy encounters shifting dynamics and reward distributions, destabilizing training and impeding convergence [26, 27].

Recent advances in large-scale language models (LLMs) present an exciting opportunity to address these challenges. LLMs excel at processing and generating meaningful representations from natural language [183–185], and have been increasingly applied to robotics for tasks like instruction grounding and code generation [186–188]. Approaches that translate natural language commands into reward functions (L2R) have shown promise [73, 75, 76]. However, prior L2R methods often target monolithic policies and lack the hierarchical structure needed to efficiently solve long-horizon, sparse reward problems.

Building on prior work in this thesis that decouples higher-level learning from lower-level changes, such as preference-based optimization in HPO (Chapter 8), which uses

DPO to derive stationary signals from human judgments, we propose LGR2 that advances this paradigm by replacing explicit preferences with natural language instructions. While HPO requires curated pairwise data, LGR2 harnesses LLMs’ semantic understanding to parameterize rewards directly from user commands (e.g., via motion descriptors and coders), inheriting HPO’s feasibility guarantees (e.g., V_L -regularized subgoals) while enabling intuitive, zero-shot task specification. This progression enables HRL for real-world human-robot interaction, where language provides rich, compositional guidance without annotation overhead.

In this work, we propose **LGR2**, a novel HRL framework that harnesses LLMs to generate *language-guided reward functions* for the higher-level policy. By converting natural language instructions into symbolic, semantically meaningful reward parameters, LGR2 decouples the higher-level reward computation from fluctuations in the lower-level behavior, thus mitigating HRL non-stationarity. To further enhance training efficiency in sparse environments, we integrate goal-conditioned hindsight experience replay (HER) [95] to densify high level rewards.

Key Contributions.

1. We introduce LGR2, an end-to-end framework employing LLM-based reward generation to guide hierarchical policies, enabling efficient learning of complex robotic tasks from natural language instructions.
2. We demonstrate that the language-guided reward relabeling scheme effectively stabilizes the higher-level policy training by achieving a stationary reward signal, thereby addressing key causes of HRL instability.
3. We incorporate HER to combat sparse rewards, significantly improving sample efficiency and generalization.
4. Through extensive experiments on challenging simulated and real-world robotic navigation and manipulation tasks, we establish that LGR2 achieves more than 55% higher

success rates over strong hierarchical and flat baselines, and achieves robust zero-shot transfer to physical robots without additional fine-tuning.

9.1 Language Guided Reward Relabeling

9.1.1 Hierarchical Setup.

We model the robotic control task as goal-conditioned Markov Decision Process (MDP) defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, r, \gamma, \mathcal{G})$, where:

- $\mathcal{S}$ is the state space,
- $\mathcal{A}$ is the action space,
- $p(s' | s, a)$ is the transition probability distribution,
- $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function,
- $\mathcal{G}$ is the goal (or goal state) space,
- $\gamma \in (0, 1)$ is the discount factor.

The policy π maps states and goals to distributions over actions. In the goal-conditioned setup, the policy is conditioned on both the current state s_t and a desired goal g_t , such that $a_t \sim \pi(\cdot | s_t, g_t)$.

We adopt a two-level hierarchical reinforcement learning (HRL) framework where control is decomposed into:

- A **higher-level policy** $\pi_H : \mathcal{S} \rightarrow \Delta(\mathcal{G})$, which selects subgoals $g_t \in \mathcal{G} \subseteq \mathcal{S}$ every k environment steps.
- A **lower-level policy** $\pi_L : \mathcal{S} \times \mathcal{G} \rightarrow \Delta(\mathcal{A})$, that conditions on the current state and the subgoal g_t to execute k primitive actions aimed at achieving g_t .

At k step intervals, the higher-level policy stores experience tuples in its replay buffer:

$\Sigma_t = (s_t, g^*, g_t, r_t^H, s_{t+k})$, where

- s_t is the current state,
- $g^* \in \mathcal{G}$ is the final user-specified goal,
- g_t is the subgoal chosen by the higher-level policy,
- $r_t^H = \sum_{i=0}^{k-1} r(s_{t+i}, a_{t+i})$ is the cumulative environment reward over the k lower-level steps,
- s_{t+k} is the next state after executing lower-level actions.

Similarly, the lower-level policy stores transitions in its replay buffer in the form: $(s_t, g_t, a_t, r_t^L, s_{t+1})$, where r_t^L is typically a reward encouraging progress towards the subgoal g_t (e.g., $r_t^L = -\mathbf{1}_{\{\|s_t - g_t\| > \varepsilon\}}$).

The hierarchical structure enables temporal abstraction, where the higher-level policy focuses on strategic subgoal selection, and the lower-level policy handles execution details over shorter time horizons.

9.1.2 Language to Rewards.

Recent work has explored translating natural language instructions into reward functions to guide RL [73–75]. These approaches typically rely on training domain-specific reward models that interpret language commands and produce corresponding reward signals. Notably, prior work [76] introduces a modular *reward translator* framework comprising two key components: a *motion descriptor*, which converts natural language instructions into structured action descriptions, and a *reward coder*, which maps these descriptions into concrete reward function parameters. This pipeline enables efficient interfacing between high-level language commands and the low-level robotic control primitives, facilitating end-to-end task specification without requiring task-specific reward engineering.

Despite significant progress in using language to specify robotic objectives, several core limitations persist in the context of complex, long-horizon tasks:

9.1.3 Challenges of Language-to-Reward (L2R) Approaches.

Existing L2R methods face three principal obstacles. First, learning a single-level policy for long-horizon tasks is fundamentally difficult due to long-term credit assignment: sparse or delayed rewards make it hard for the agent to attribute success or failure to specific actions taken many steps prior. Second, reward functions automatically synthesized from natural language instructions are often highly sparse, providing insufficient learning signal for the lower-level policy to make meaningful progress, especially in tasks requiring intricate sequences of behaviors. Third, many prior approaches rely on either pre-defined or manually-engineered libraries of control primitives to bridge the gap between abstract language instructions and executable actions. This not only demands significant expert knowledge and engineering effort, but also reduces scalability and adaptability to new tasks or environments.

These intertwined challenges: (i) lack of temporal abstraction and compositionality in L2R methods, (ii) reward sparsity, (iii) the inflexibility of manual skill libraries, and (iv) non-stationarity in hierarchical RL, motivate the need for new frameworks that combine the semantic richness of language guidance with scalable, stable hierarchical learning for complex robotic control.

Now, we discuss our approach LGR2 which proposes *language-guided reward relabeling* to produce a stationary and semantically meaningful high-level reward function independent of the evolving lower-level policy, thus mitigating non-stationarity issue in HRL.

9.1.4 Reward Parameter Generation

Given a natural language instruction $\mathcal{L}$ provided by the user, a Large Language Model (LLM) is employed to generate a structured representation of the task and subsequently

translate it into reward function parameters. Inspired from L2R framework [76], this is implemented via a two-stage pipeline using a motion descriptor and a reward coder module. This established pipeline operates as follows:

- **Motion Descriptor Module:** Given a user’s natural language instruction $\mathcal{L}$, this module uses large language models (LLMs) to generate a structured task description $d = \mathcal{M}_{\text{desc}}(\mathcal{L})$ in templated natural language. This step transforms ambiguous instructions into clear, canonical representations that facilitate downstream processing [76].
- **Reward Coder Module:** Conditioned on the structured motion descriptor d , the reward coder uses the LLM’s code generation capabilities to output parameters $\phi = \mathcal{M}_{\text{code}}(d)$ for a symbolic, goal-conditioned reward function. The coder produces explicit, executable code (or interpretable parameter sets) that define the reward evaluation logic for the specified skill.

Formally, the resulting language-conditioned reward function is a mapping

$$r_\phi : \mathcal{S} \times \mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R},$$

where for each triple (s, g^*, g) ,

- $s \in \mathcal{S}$ represents the current environment state (or a relevant subset of state features),
- $g^* \in \mathcal{G}$ denotes the final task goal specified by the user,
- $g \in \mathcal{G}$ is a candidate subgoal proposed by the higher-level policy.

The higher level reward function r^H evaluates

$$r^H = r_\phi(s, g^*, g) = f(s, g^*, g; \phi),$$

where f is a parameterized function with fixed parameters ϕ generated by the LLM-based reward coder. Note that the reward function r_ϕ depends exclusively on the current

state and goal variables, and is independent of the low-level policy. This decoupling effectively mitigates non-stationarity in HRL by providing a stable, consistent reward signal. Consequently, reward assignment remains stationary, semantically meaningful, and highly modular, which are essential for reliable and efficient hierarchical learning in complex, long-horizon robotic control tasks.

9.1.5 Relabeling of High-Level Rewards

Each stored transition in the higher-level replay buffer,

$$\Sigma_t = (s_t, g^*, g_t, r_t^H, s_{t+k}), \quad (9.1)$$

is relabeled as:

$$\tilde{\Sigma}_t = (s_t, g^*, g_t, r_t^\varphi, s_{t+k}), \quad (9.2)$$

where the *language-guided reward* is defined as

$$r_t^\varphi := r_\varphi(s_t, g^*, g_t). \quad (9.3)$$

Since r_φ does not depend on the lower-level policy π_L or the agent’s trajectories, it remains *stationary* across training iterations. This removes reward drift prevalent in off-policy HRL, thereby stabilizing the learning process for the higher-level policy. Consequently, the relabeled buffer $\mathcal{D}_H = \{\tilde{\Sigma}_t\}$ is used for training the higher-level policy.

9.1.6 Addressing Reward Sparsity with HER

In environments with sparse rewards, the high-level reward signal, r_φ , can often be too sparse to produce a meaningful reward signal. To overcome this challenge, **LGR2** incorporates goal-conditioned hindsight experience replay [95] to densify the reward signal.

Concretely, for each relabeled high-level transition $\tilde{\Sigma}_t$, we sample alternative goals $\hat{g}$ from states encountered within the same trajectory, i.e.,

$$\hat{g} \in \{s_{t+j} \mid 0 \leq j \leq k\}.$$

We then relabel the transition as

$$\tilde{\Sigma}_t^{\text{HER}} = (s_t, \hat{g}, g_t, r_\varphi(s_t, \hat{g}, g_t), s_{t+k}),$$

which is added to the replay buffer $\mathcal{D}^H$. This hindsight relabeling increases the likelihood of encountering positive rewards in the replay buffer, thereby enhancing sample efficiency. The boosted frequency of informative reward signals encourages more effective exploration and leads to improved training stability in sparse-reward, long-horizon tasks.

9.1.7 Training Objectives

Both the lower-level and higher-level policies are trained using the Soft Actor-Critic [114] framework with separate critic networks, adapted to their respective roles in the hierarchical setup.

9.1.7.1 Lower-Level Policy Loss

The lower-level policy π_L aims to achieve the subgoals issued by the higher-level policy within k environment steps. Its training relies on a standard SAC critic update based on the replay buffer $\mathcal{D}_L$:

$$J_L = \mathbb{E}_{(s,g,a,r,s') \sim \mathcal{D}_L} \left[\left(r + \gamma Q_\theta^L(s', g) - Q_\theta^L(s, g, a) \right)^2 \right], \quad (9.4)$$

Algorithm 7 LGR2

```

1: Initialize higher level replay buffer  $R^H = \{\}$  and lower level replay buffer  $R^L = \{\}$ 
2: for  $i = 1 \dots N$  do
3:   // Collect experience using  $\pi^H$  and  $\pi^L$  and store transitions in  $R^H$  and  $R^L$ 
4:   for each timestep  $t$  do
5:      $d^H \leftarrow d^H \cup \{(s_t, g^*, g_t, \sum_{i=t}^{t+k-1} r_i, s_{t+k-1})\}$ 
6:      $d^L \leftarrow d^L \cup \{(s_t, g_t, a_t, r_t, s_{t+1})\}$ 
7:      $R^H \leftarrow R^H \cup d^H$ 
8:      $R^L \leftarrow R^L \cup d^L$ 
9:   // Sample and relabel higher-level behaviour trajectories
10:  for  $i = 1 \dots M$  do
11:     $\sigma \sim R^H$  where  $\sigma = \{(s_t, g^*, g_t, \sum_{i=t}^{t+k-1} r_i, s_{t+k-1})\}_{t=1}^{n-1}$ 
12:    // Relabel the reward using language-guided reward  $r_\phi$ , and store the transition
    in  $R^H$ 
13:     $R^H \leftarrow R^H \cup \{(s_t, g^*, g_t, r_\phi, s_{t+k-1})\}_{t=1}^{n-1}$ 
14:    // Sample a set of additional goals for HER ( $\hat{G}$ )
15:    for  $\hat{g} \in \hat{G}$  do
16:      // Relabel  $g$  by  $\hat{g}$  and  $r_\phi$  by  $\hat{r}_\phi$  in  $\sigma$  to generate  $\hat{\sigma} = \{(s_t, \hat{g}, g_t, \hat{r}_\phi, s_{t+k-1})\}_{t=1}^{n-1}$ 
17:      Store in replay buffer  $R^H \leftarrow R^H \cup \hat{\sigma}$ 
18:    // Policy Learning
19:    for each gradient step do
20:      Sample  $\{(\sigma_j)\}_{j=1}^m$  from  $R^H$ 
21:      Sample  $\{(\delta_j)\}_{j=1}^m$  from  $R^L$ 
22:      Optimize higher policy  $\pi^H$  using SAC
23:      Optimize lower policy  $\pi^L$  using SAC

```

where $s, s' \in \mathcal{S}$ are states, $g \in \mathcal{G}$ is the current goal or subgoal, $a \in \mathcal{A}$ is the action, r is the reward signal for the lower-level policy, Q_θ^L is the current lower-level critic parameterized by θ , and Q_θ^L is the target critic network.

9.1.7.2 Higher-Level Policy Loss

The higher-level policy π_H is trained using relabeled, stationary rewards derived from language-guided reward functions (r_ϕ) and the higher-level replay buffer $\tilde{\mathcal{D}}_H$:

$$J_H = \mathbb{E}_{(s, g^*, g, r_\phi, s') \sim \tilde{\mathcal{D}}_H} \left[(r_\phi + \gamma^k Q_\omega^H(s', g') - Q_\omega^H(s, g^*, g))^2 \right], \quad (9.5)$$

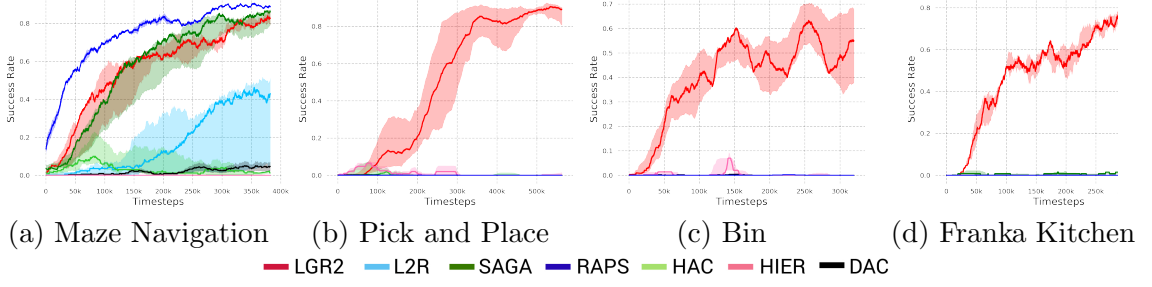


FIGURE 9.2: **Success rate comparison** This figure compares the success rate performances on four sparse maze navigation and robotic manipulation environments. The solid line and shaded regions represent the mean and standard deviation, across 5 seeds. We compare our approach LGR2 against multiple baselines. LGR2 shows impressive performance and significantly outperforms the baselines.

where $s, s' \in \mathcal{S}$, $g^* \in \mathcal{G}$ is the final goal, $g \in \mathcal{G}$ is the current subgoal, r_φ is the stationary, language-guided reward, Q_ω^H and $Q_{\bar{\omega}}^H$ are the current and target higher-level critics parameterized by ω and $\bar{\omega}$ respectively, and $g' \sim \pi_H(s')$ is the next subgoal sampled from the higher-level policy.

The language-guided reward r_φ is invariant with respect to the evolving lower-level policy, thus enhancing training stability. We provide the algorithm in Algorithm 7.

9.2 Experiments

In this section, we investigate the following questions:

- How does LGR2 perform on complex, sparse-reward robotic navigation and manipulation tasks compared to prior hierarchical and single-level baselines?
- Can language-guided reward relabeling generate better rewards than alternative baselines?
- Does language-guided reward relabeling outperform standard HRL with hindsight relabeling?
- Can LGR2 mitigate non-stationarity in HRL?

- What is the contribution of hindsight experience relabeling (HER) in improving performance?
- What are the qualitative insights into LGR2’s improved performance?
- Can LGR2 policies transfer to real-world robots?

Implementation details. We evaluate LGR2 on four continuous sparse reward robotic tasks: maze navigation, pick and place, bin, and franka kitchen [189]. In the maze task, intermediate waypoints serve as subgoals; for more complex tasks like franka kitchen, full intermediate states are used as subgoals. The policy networks consist of three fully connected layers with 512 units each, trained using SAC [114] and Adam optimizer [115]. We carefully tune hyperparameters via grid search across all baselines to ensure fair comparison. For challenging tasks such as pick and place and kitchen, we incorporate a single demonstration and an imitation learning objective at the lower level; no demonstrations are used for maze navigation to maintain consistent evaluation.

Sparse Rewards and Challenges of Goal-Conditioning. Although some tasks may appear simple, all our evaluation environments feature inherently sparse reward structures, requiring agents to engage in extensive exploration before encountering any positive feedback. This sparse reward setting significantly amplifies the difficulty of the tasks. Additionally, we adopt a goal-conditioned RL framework with randomly generated initial positions and final goals, further increasing the complexity by demanding generalization across diverse objectives. These factors combined lead to markedly challenging training conditions, which explain why most baseline methods perform poorly in our experiments. The sparse and delayed nature of the rewards, coupled with high variability in goals, makes efficient credit assignment and exploration extremely difficult for flat or naive hierarchical policies. In contrast, our hierarchical approach benefits from temporal abstraction, and is able to significantly outperform the baselines in most tasks.

Computational Overhead of LLM-Integrated Reward Generation. While LGR2 leverages LLMs for expressive, flexible reward generation, the main computational cost, ie

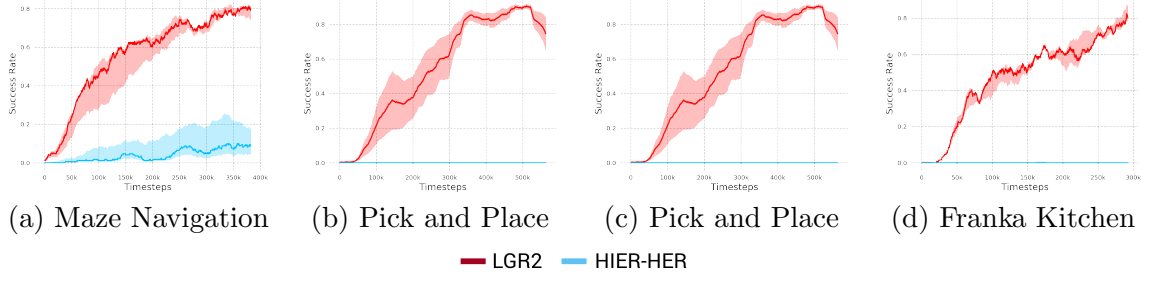


FIGURE 9.3: **Comparison against HRL+HER** This figure compares the cumulative rewards on four sparse maze navigation and robotic manipulation environments. We compare our approach LGR2 against the hierarchical baseline HIER-HER. LGR2 outperforms this baseline, which shows the efficacy of language-guided reward generation in LGR2.

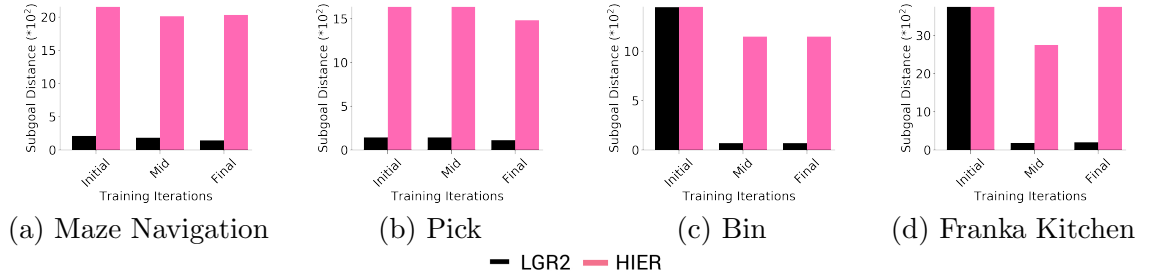


FIGURE 9.4: **Non-stationarity metric comparison** This figure compares the average distance metric between the subgoals predicted by the higher level policy and the states achieved by the lower level policy during training. (The columns represent Initial: when training begins, Mid: half-way during training, Final: when training ends, e.g. since maze navigation is trained for $3.8\text{E}5$ timesteps, the values are Initial: iteration 1, Mid: iteration $1.9\text{E}5$, and Final: iteration $3.8\text{E}5$). As can be seen, LGR2 consistently generates efficient and achievable subgoals, thereby mitigating non-stationarity in HRL.

translating natural language instructions into structured motion descriptors and reward-coder parameters via LLM calls, is incurred *once per task specification*, before interaction and training. This offline design prevents LLM calls from bottlenecking RL, as later reward evaluations are lightweight, involving only fast code execution. Memory impact is minimal since only the reward code or parameters are stored, not full LLM outputs. LGR2 maintains sample efficiency and is practical for both simulation and real robotic applications.

How does LGR2 perform on complex control tasks compared to prior baselines?

We compare LGR2 with hierarchical and flat baselines (Figure 9.2); curves show mean $\pm$ std over 5 seeds.

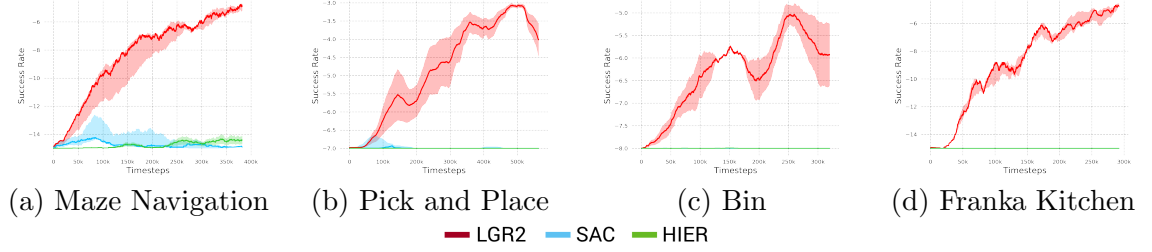


FIGURE 9.5: **Reward comparison** This figure compares the average rewards per episode metric on four sparse maze navigation and robotic manipulation environments. The solid line and shaded regions represent the mean and standard deviation, across 5 seeds. We compare our approach LGR2 against HAC and HIER. LGR2 achieves the highest reward, which shows that our language-guided reward relabeling based approach generates high rewards, compared to other hierarchical baselines.

L2R. The original L2R approach [76] translates natural language instructions into reward function parameters through a reward translator module and optimizes them using a Mujoco MPC controller [77]. To ensure fair comparison, our L2R baseline implements the same reward translator module but uses a Soft Actor-Critic (SAC) [114] RL agent in place of the MPC controller. While the reward translator outputs reward parameters that SAC then optimizes, this single-level baseline achieves some progress in the maze task but fails on the harder, sparse-reward domains. In contrast, LGR2 consistently outperforms L2R (see Figure 9.2), primarily because its hierarchical structure enables temporal abstraction and decomposes difficult tasks into more manageable subgoals, crucial for solving complex robotic control tasks.

SAGA. We compare LGR2 with SAGA [180], a hierarchical approach that employs a state-conditioned discriminator to align high-level subgoals with the evolving low-level policy. While SAGA achieves strong results on the maze domain, its performance deteriorates in more challenging benchmarks, whereas LGR2 delivers significantly better outcomes. This performance gap highlights that, while SAGA attempts to address non-stationarity, it is more susceptible to instability in complex, long-horizon settings. LGR2’s advantage stems from its use of robust language-guided reward relabeling, which provides a stationary and semantically meaningful supervisory signal, thus stabilizing hierarchical training across diverse tasks.

RAPS. The RAPS baseline [116] uses a library of predefined robot action primitives

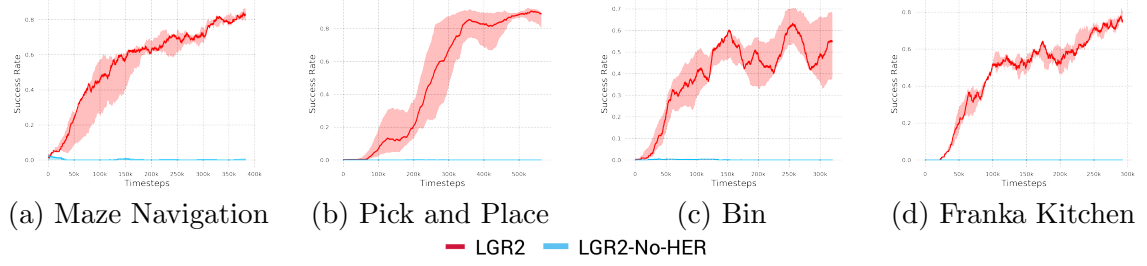


FIGURE 9.6: **Hindsight Relabeling ablation** This figure compares the performance of LGR2 with LGR2-No-HER ablation (LGR2 without hindsight relabeling). The plots clearly demonstrate that HER is crucial for good performance in all the tasks.

controlled by the higher-level policy. While RAPS performs well on simpler maze tasks, it struggles on all other harder tasks. This suggests methods relying on fixed primitives lack flexibility for complex objectives. In contrast, LGR2 autonomously discovers effective subgoal decompositions and adapts dynamically, achieving strong results on multi-step, long-horizon tasks.

HAC. HAC [95] attempts to mitigate non-stationarity by assuming an optimal lower-level policy and enhances the higher-level replay buffer using subgoal relabeling. Despite this, our experiments show LGR2 surpasses HAC (see Figure 9.2). We believe that this is because LGR2’s language-guided reward relabeling generates stable, goal-aligned rewards for the higher-level policy without requiring the unrealistic assumption of an optimal lower-level primitive, thus providing a more practical and robust mechanism for addressing non-stationarity.

HIER. The HIER baseline is a standard hierarchical SAC model where the higher-level reward is the cumulative sum of environment rewards across k steps. LGR2 achieves much higher performance than HIER, underscoring the critical role of language-guided reward generation in reducing non-stationarity and resulting in stable hierarchical learning.

DAC. Finally, we also consider DAC baseline with access to a single demonstration. DAC fails to show any significant progress, as seen in Figure 9.2, which shows that LGR2 outperforms single-level baselines with access to privileged information like expert demonstrations.

Can language-guided reward relabeling achieve better rewards than alternative baselines?

To evaluate the effectiveness of language-guided reward relabeling in densifying rewards and promoting exploration, we compare the *average reward per episode* achieved by LGR2 with two hierarchical baselines: HAC and HIER. As shown in Figure 9.2, LGR2 consistently achieves substantially higher average rewards throughout training, demonstrating its ability to generate richer rewards. This reflects the benefit of using language-derived, re-labeled high-level rewards, which provide semantically meaningful feedback even in highly sparse environments. In contrast, HAC and HIER both suffer from limited reward density, leading to slower learning and inefficient exploration. These results underscore that our language-guided reward relabeling is key to producing informative rewards and improve performance in complex, sparse-reward robotic tasks.

Does language-guided reward relabeling outperform standard HRL with hindsight relabeling?

We assess the isolated contribution of language-guided reward relabeling by comparing LGR2 with the ablation *HIER-HER*, a hierarchical RL baseline that uses HER but relies solely on environment signals for high-level rewards, with no language guidance. This setup retains all elements of LGR2 except language-driven reward relabeling. As shown in Figure 9.3, LGR2 consistently outperforms HIER-HER across all tasks, achieving greater learning efficiency and higher success rates in complex sparse-reward environments. These improvements highlight that while HER effectively densifies feedback, the core gains of LGR2 stem from the ability of language-guided relabeling to provide a stationary, semantically aligned supervisory signal for the high-level policy. This stabilizes hierarchical learning and leads to improved performance over baselines.

Can LGR2 mitigate non-stationarity in HRL?

We show LGR2’s ability to mitigate non-stationarity in HRL in Figure 9.4. To quantify this, we compare LGR2 to the *HIER* baseline by measuring the average distance between subgoals predicted by the higher-level policy and the final states reached by the lower-level

policy at different training stages (Initial: start of training, Mid: halfway point, Final: end of training). Lower distance values indicate that the higher-level policy generates subgoals that are feasible and well-aligned with the abilities of the evolving lower-level policy, leading to more consistent goal achievement and reduced non-stationarity. As shown in Figure 9.4, LGR2 consistently achieves lower average distances across training, validating that it produces achievable subgoals and robustly reduces non-stationarity compared to the baseline.

What is the contribution of hindsight experience relabeling (HER) in improving performance?

Further, we analyse the importance of hindsight experience replay (HER) [95] for densifying rewards in sparse reward tasks. To this end, we compare LGR2 with LGR2-No-HER baseline, which is LGR2 baseline without HER. As seen in Figure 9.6, HER is crucial for good performance, since language-guided rewards are too sparse to generate any meaningful reward signal.

What are the qualitative insights into LGR2’s improved performance?

Beyond quantitative learning curves, several qualitative aspects highlight the superiority of our **LGR2** framework over competing approaches:

1. **Stable and Consistent Subgoal Generation:** Unlike conventional HRL methods, which suffer from *non-stationarity* caused by evolving lower-level policies, LGR2 employs language-guided reward relabeling to generate *stationary* and *semantically meaningful* rewards. This leads to the higher-level policy consistently proposing *achievable* and *task-aligned* subgoals throughout training, as evidenced by the decreasing distance between subgoals and achieved states over time (Figure 9.4). This stable subgoal generation fosters more reliable exploration and efficient credit assignment at the high level.

2. **Decoupling Reward Signal from Lower-Level Drift:** Competing hierarchical methods rely on environment-derived or evolving low-level policy-dependent rewards, often resulting in shifting reward distributions that destabilize learning. Our approach uniquely decouples the reward function using a precomputed, language-conditioned reward parameterization derived from large language models (LLMs). This decoupling directly mitigates the non-stationarity problem, enabling *robust convergence* and *efficient hierarchical credit assignment* even in sparse reward settings.
3. **Effective Use of Temporal Abstraction via Subgoals:** The hierarchical structure of LGR2 separates long-term planning from short-horizon execution. While flat or single-level baselines struggle to solve multi-step manipulation and navigation tasks due to long-term credit assignment challenges, LGR2’s higher-level policy learns to break down complex tasks into manageable subgoals. These subgoals serve as intermediate milestones that are easier for the lower-level policy to accomplish, facilitating *progressive updates* towards the overall objective.
4. **Enhanced Sample Efficiency and Generalization:** Integrating goal-conditioned hindsight experience replay (HER) with language-guided relabeling densifies the learning signal, particularly in sparse reward environments. This allows LGR2 to learn faster and generalize better across varied task instances, outperforming competing methods that rely solely on environment rewards or handcrafted skill libraries.
5. **Robust Zero-Shot Transfer to Real Robots:** While many competing methods require extensive fine-tuning or demonstrations on real hardware, LGR2 policies trained in simulation demonstrate *strong zero-shot transfer* to real robotic systems.

Summary These qualitative observations reveal that LGR2’s advantage arises from its principled use of language-guided reward relabeling to stabilize hierarchical learning, its effective temporal abstraction through subgoaling, and its ability to generate semantically grounded, stationary rewards that are robust to lower-level policy changes. As a result, LGR2 solves complex, long-horizon robotic tasks with higher success rates and greater training stability compared to prior hierarchical and flat RL baselines.

Can LGR2 policies transfer to real-world robots?

We conducted real-world experiments on pick-and-place and bin tasks, using a RealSense D435 depth camera to track the robotic arm, block, and bin positions. The robot was controlled via the manufacturer’s Python SDK at 20 Hz ($\Delta t = 50$ ms), with episodes lasting about 40 seconds. Policies were trained in simulation and then deployed on the robot; due to challenges in precise velocity control, we used small fixed velocities, achieving good performance. Across 5 sets of 10 trials each, LGR2 attained average success rates of 60% and 50%, with variances 0.07 and 0.03 for pick-and-place and bin tasks, respectively. The best baseline, L2R, failed to show significant progress.

9.3 Full Prompts

We provide detailed prompts for motion descriptor and reward translator for all tasks.

Maze navigation Environment:**Task Description:**

Navigate a 7-DOF robotic arm gripper through randomly generated four-room mazes to reach a specified goal position.

Example Instructions:

1. “Navigate the gripper from the starting position to the goal while avoiding walls.”
2. “Move through the maze following the shortest collision-free path to the goal.”
3. “Guide the end-effector to the target location via the available passages.”

Motion Descriptor Prompt for Maze Navigation environment

Generate a random position for object on the table following the description and rules.

[Description]

1. There is a table which can be represented as a matrix of (num_1, num_2).
2. Generate walls within the table by choosing a random row and random column and blocking all of (num_1+num_2) cells.
3. Generate four random cells from the (num_1+num_2) marked as walls. Remove the blocks from these cells and mark them as gates.
4. Generate final position for CHOICE:[cuboid,apple,ball] with (x,y) co-ordinates between (num_4, num_5) and height is at table height in the bottom right corner.

[Rules]

1. The robot is a 7-DOF robotic arm gripper.
2. The height of the table is table_height=0.42 cm.
3. If you see num_1 replace it with an integer within 10 and 20.
4. If you see num_2 replace it with an integer within 10 and 20.
5. If you see phrases like CHOICE: [choice1, choice2, ...], it means you should replace the entire phrase with one of the choices listed.
6. The final position cannot coincide with gates, walls or starting position location.
7. The starting position of the location is (1,3)

Reward Generator Prompt for Maze Navigation maze environment

We have a description of a robot's motion and we want you to turn that into the corresponding program with following functions:

```
def reset_environment()
```

```
def set_Gripper_Pos(x_pos, y_pos, z_pos)
```

x_pos: position of x-coordinate of the gripper of robot arm.

y_pos: position of y-coordinate of the gripper of robot arm.

z_pos: position of z-coordinate (height) of the gripper of robot arm.

```
generate_maze()
```

```
do_simulation()
```

Example answer code:

```
import numpy as np reset_environment()
```

```
#This is a new task so reset environment else we do not need it.
```

```
set_Gripper_Pos(3.0,2.0,0.56)
```

```
set_Gripper_Pos(4.45,3.56,0.48)
```

```
set_Gripper_Pos(6.85,7.36,0.64)
```

```
generate_maze()
```

```
#generate maze with all the constraints
```

```
do_simulation()
```

```
#run the simulation
```

Pick and Place Environment

Task Description: Pick up a randomly placed object and transport it to a specified goal location using a 7-DOF robotic manipulator.

Example Instructions:

1. “Pick up the red cube and place it at the designated target location.”
2. “Grasp the cylindrical object and transport it to the goal position.”
3. “Carefully lift the sphere and set it down at the specified coordinates.”

Motion Descriptor Prompt for Pick and Place Environment

Generate a random position for an object within the table following the description and rules.

[Description]

1. The table can be represented as a matrix of (num_1, num_2) and height (num_3).
2. Generate final position for the object like CHOICE:[cuboid,apple,ball] with the (x,y) co-ordinates between (num_4,num_5) and height (num_6).

[Rules]:

1. The robot is a 7-DOF robotic arm gripper.
2. The height of the table is table_height=0.42 cm.
3. The max height the arm can reach is max_height=0.66 cm.
4. If you see num_1 replace it with an integer within 10 and 20.
5. If you see num_2 replace it with an integer within 10 and 20.
6. If you see num_3 replace it with 0.42.
7. If you see phrases like CHOICE: [choice1, choice2, ...], it means you should replace the entire phrase with one of the choices listed.
8. Please remember that there is an object on the table.
9. The block is light enough for the robot to pick up and hold in the air for a long time, like 4 seconds.

Reward Generator Prompt for Pick and Place Environment

We have a description of a robot's motion and we want you to turn that into the corresponding program with following functions:

```
def reset_environment()
```

```
def set_Gripper_Pos(x_pos, y_pos, z_pos)
```

x_pos: position of x-coordinate of the gripper of robot arm.

y_pos: position of y-coordinate of the gripper of robot arm.

z_pos: position of z-coordinate (height) of the gripper of robot arm.

```
def generate_Object_Pos()
```

```
def do_simulation()
```

Example answer code:

```
import numpy as np
```

```
reset_environment()
```

```
#This is a new task so reset environment else we do not need it.
```

```
set_Gripper_Pos(3.0,2.0,0.56)
```

```
set_Gripper_Pos(4.45,3.56,0.48)
```

```
set_Gripper_Pos(6.85,7.36,0.64)
```

```
do_simulation()
```

```
#run the simulation
```

Bin Environment

Task Description: Pick up an object and place it inside a designated bin positioned on the workspace.

Example Instructions:

1. “Pick up the object and carefully place it inside the collection bin.”
2. “Grasp the item and insert it into the designated container.”
3. “Transport the object to the bin and ensure proper placement.”

Motion Descriptor Prompt for Bin Environment

Generate a random position for bin and an object within the bin following the description and rules.

[Description]

1. The table can be represented as a matrix of (num_1, num_2) and height (num_3).
2. There is a bin on the table.
3. Generate a random position for the bin within the table.
4. Generate a final position (x,y) for placing the CHOICE:[cuboid,apple,ball] object within the bin.

[Rules]

1. The robot is a 7-DOF robotic arm gripper.
2. The height of the table is table_height=0.42 cm.
3. The max height the arm can reach is max_height=0.66 cm.
4. If you see num_1 or num_2 replace it with an integer within 10 and 20.
5. If you see num_3 replace it with 0.42.
6. The bin has to be completely within the table.
7. The height, width and length of the table are 0.1 cm.
8. The final position of the object should be continuous and at centre of the bin.
9. If you see phrases like CHOICE: [choice1, choice2, ...], then replace the entire phrase with one of the choices.
10. Please remember that there is always a bin on the table.
11. The object is light enough for the robot to pick up and hold in the air for a long time, like 4 seconds.

Reward generator prompt for Bin environment

We have a description of a robot's motion and we want you to turn that into the corresponding program with following functions:

```
def reset_environment()
```

```
def set_Gripper_Pos(x_pos, y_pos, z_pos)
```

x_pos: position of x-coordinate of the gripper of robot arm.

y_pos: position of y-coordinate of the gripper of robot arm.

z_pos: position of z-coordinate (height) of the gripper of robot arm.

```
def generate_Bin()
```

```
def do_simulation()
```

Example answer code:

```
import numpy as np
```

```
reset_environment()
```

```
#This is a new task so reset environment else we do not need it.
```

```
set_Gripper_Pos(3.0,2.0,0.56)
```

```
set_Gripper_Pos(4.45,3.56,0.48)
```

```
set_Gripper_Pos(6.85,7.36,0.64)
```

```
generate_Bin()
```

```
#generate bin with all the constraints
```

```
do_simulation()
```

```
#run the simulation
```

Franka Kitchen Environment

Task Description: Perform complex multi-step kitchen tasks involving appliance manipulation such as opening microwaves and operating gas knobs.

Example Instructions:

1. “Open the microwave door and then turn on the gas burner.”
2. “Operate the kitchen appliances in the specified sequence.”
3. “Manipulate the microwave and gas controls to complete the cooking preparation.”

Motion descriptor prompt for Franka kitchen environment

We want you to generate a random position for the door of a microwave and gas-knob following the description and rules.

[Description]

1. There is a microwave and gas knob.
2. Open the microwave door to a certain point num_1.
3. Turn the gas knob to co-ordinates (num_2, num_3).

[Rules]

1. The robot is a 7-DOF robotic arm gripper.
2. The height of the table is table_height=0.42 cm.
3. The max height the arm can reach is max_height=0.66 cm.
4. If you see num_1 replace it with a continuous number -0.75.
5. If you see num_2 replace it with a continuous number -0.88.
6. If you see num_2 replace it with a continuous number -0.01.

Reward generator prompt for Franka kitchen environment

We have a description of a robot's motion and we want you to turn that into the corresponding program with following functions:

```
def reset_environment()
```

```
def set_microwave_Pos(joint_1_Pos)
```

joint_1_Pos: Position of joint to determine whether microwave door is open or not.

```
def set_gas_Knob_Pos(joint_1_Pos, joint_2_Pos)
```

joint_1_Pos: x co-ordinate of the gas knob.

joint_2_pos: y co-ordinate of the gas knob.

```
def do_simulation()
```

Example answer code:

```
import numpy as np
```

```
reset_environment()
```

```
#This is a new task so reset environment else we do not need it.
```

```
set_microwave_Pos(-0.75)
```

```
set_microwave_Pos(-0.55)
```

```
set_gas_Knob_Pos(-0.88, -0.01)
```

```
set_gas_Knob_Pos(-0.75, -0.1)
```

```
do_simulation()
```

```
#run the simulation
```

9.3.1 Qualitative Behavioral Explanations

Maze Navigation Environment

The hierarchical policy learns to decompose navigation into waypoint-based subgoals. The higher-level policy identifies key intermediate positions that avoid walls and utilize gates, while the lower-level policy executes smooth trajectories between waypoints. The agent exhibits emergent path-planning behavior, automatically discovering efficient routes through the four-room structure without explicit path-planning algorithms.

Pick and Place Environment

The system develops a natural three-phase manipulation strategy: (1) approach the object, (2) grasp the object, (3) lift and transport to target location. The higher-level policy learns to predict intermediate gripper positions that facilitate successful grasping, while the lower-level policy handles fine motor control for object manipulation. Recovery behaviors emerge when initial grasps fail.

Bin Environment

The agent learns a sophisticated insertion strategy involving pre-positioning above the bin, careful alignment, and controlled lowering. The higher-level policy predicts subgoals that account for the bin’s spatial constraints, while the lower-level policy develops precise positioning skills. The system exhibits adaptive behavior when objects are initially misaligned with the bin opening.

Franka Kitchen Environment

The higher-level policy sequences subgoals for microwave door manipulation followed by gas knob operation, while the lower-level policy masters the precise joint control required for each appliance. The system develops robust error recovery when appliance states don’t match expectations.

9.4 Sample Codes

In this section, we show sampled responses for all four environments mentioned in this work for a better understanding of the LGR2 method. We sample 2-3 responses for each

method to show both successful and failed results.

Sample Code for Maze Navigation environment Here, we provide the sample code for the Maze Navigation environment generated by GPT-3.5.

```
import numpy as np

def reset_environment():

    print("Environment reset.")

def set_Gripper_Pos(x_pos, y_pos, z_pos):

    print(f"Setting gripper position to x: x_pos, y: y_pos, z: z_pos.")

def generate_maze():

    print("Generating maze...")

def do_simulation():

    print("Running simulation...")

reset_environment()

set_Gripper_Pos(3.0, 2.0, 0.56)

set_Gripper_Pos(4.45, 3.56, 0.48)

set_Gripper_Pos(6.85, 7.36, 0.64)

generate_maze()

do_simulation()
```

Sample Code for Pick and Place environment

Here, we provide the sample code for the Pick and Place environment generated by GPT-3.5.

```
import numpy as np

def reset_environment():

    print("Environment reset.")

def set_Gripper_Pos(x_pos, y_pos, z_pos):

    print(f"Setting gripper position to x: x_pos, y: y_pos, z: z_pos.")

def generate_Object_Pos(): print("Generating object position.")

def do_simulation():

    print("Simulation started.")

    reset_environment()

    set_Gripper_Pos(3.0, 2.0, 0.56)

    set_Gripper_Pos(4.45, 3.56, 0.48)

    set_Gripper_Pos(6.85, 7.36, 0.64)

    generate_Object_Pos()

    do_simulation()
```

Sample Code for Bin environment

Here, we provide the sample code for the Bin environment generated by GPT-3.5.

```
import numpy as np

def reset_environment():

    print("Environment reset.")

def set_Gripper_Pos(x_pos, y_pos, z_pos):

    print(f"Setting gripper position to x: x_pos, y: y_pos, z: z_pos.")

def generate_Bin():

    print("Generating bin with all constraints.")

def do_simulation():

    print("Simulation started.")

reset_environment()

set_Gripper_Pos(3.0, 2.0, 0.56)

set_Gripper_Pos(4.45, 3.56, 0.48)

set_Gripper_Pos(6.85, 7.36, 0.64)

generate_Bin()

do_simulation()
```

Sample Code for Franka kitchen environment

Here, we provide the sample code for the Franka kitchen environment generated by GPT-3.5.

```
import numpy as np

def reset_environment():

    print("Environment reset")

def set_microwave_Pos(joint_1_Pos):

    print(f"Setting microwave door position to joint_1_Pos.")

def set_gas_Knob_Pos(joint_1_Pos, joint_2_Pos):

    print(f"Setting gas knob position to x: joint_1_Pos, y: joint_2_Pos.")

def do_simulation():

    print("Simulation started.")

reset_environment()

set_microwave_Pos(-0.75)

set_microwave_Pos(-0.55)

set_gas_Knob_Pos(-0.88, -0.01)

set_gas_Knob_Pos(-0.75, -0.1)

do_simulation()
```

9.5 Related Work

The development of effective hierarchies of policies has attracted considerable research interest in the reinforcement learning (RL) community[12, 22, 57, 97]. Prior works have

attempted to pre-train policies on related tasks to develop behaviour skill priors [190, 191], followed by fine-tuning using RL. However, these approaches may struggle to generalize when there is a distributional shift between the initial and target task distributions, and may also fail to learn effective policies due to sub-optimal priors. Some methods rely on hand-designed action primitives [116, 192] to determine how to solve sub-tasks and subsequently predict what to do by selecting the appropriate primitives, which can be labour-intensive, limiting the applicability of such approaches to complex real-world tasks. LGR2 avoids these limitations by learning hierarchical policies in parallel, thus eliminating the need for hand-designed action primitives and enhancing the ability to handle more complex tasks.

Language to Actions The earliest work of this paradigm studied mapping templated language to controllers with temporal logic [193] or learning a parser to motion primitives [194]. Recent studies focus on utilizing end-to-end models that produce actions based on natural language descriptions. [195] uses natural language descriptions for instructions for navigation methods of robots. However, they suffer from the assumption of low-dimensional actions navigating from one graph node to another [195, 196]. Utilizing latent embeddings of language commands as multiple input text is a commonly used approach to extend the end-to-end approaches for navigation manipulation. These embeddings are either trained with behavioural cloning [197–199], offline RL [200], goal-conditioned RL [201], or in a shared autonomy paradigm [202] to translate the natural language descriptions into robot navigation methods. End-to-end policies require a large amount of training data. [76] propose a less data-centric approach where an optimal controller generates low-level actions. In contrast, we employ an HRL based approach that leverages temporal abstraction to induce training efficiency.

Language to Code Large Language Models (LLMs) like Llama [203], GPT-4 [204], and Gemini [205] have been extensively used for code generation both in and outside the robotics context [206–208]. These models are highly efficient in solving coding competition questions [209], drawing figures [210], generating policies to solve 2D tasks [211], and

complex instruction tasks [212]. In this work, we use LLMs to generate language-guided higher-level rewards.

Language to Rewards Several works have explored the idea of translating natural language instructions to rewards [73, 75, 213–217]. Prior works have gone into training domain-specific reward models to map language instructions to reward values [73, 213, 214] or constraints [73]. These models can perform efficiently for challenging robotic tasks such as object pushing [73] and drawer opening [214] but are hindered by the requirement of a large amount of language-labelled data. Recent research has focused on employing LLMs as a reward function to infer user intents in negotiation games or collaborative human-AI interaction games [75, 216]. All these works impeded utilizing the LLMs to assign reward values during RL training. Nevertheless, in training RL rules, these works get reward values of rollouts, necessitating a high volume of inquiries to LLMs. AutoRL [218] proposed an automated parameterization of reward functions, but they did not provide a language interface. We leverage reward generated from of language instructions to mitigate non-stationarity in HRL, and thus propose an effective framework for translating language instructions to robot actions.

9.6 Discussion

Limitations and future work Although we demonstrate that LGR2 is able to solve complex robotic control tasks through natural language instructions, it faces certain limitations. Firstly, designing prompts for the reward translator module requires additional manual effort. Alternatively, if we neglect designing the prompts, the LLMs might hallucinate the parameters for reward function. An interesting direction for future work is to automatically generate the prompts, or even disregard the prompts altogether. We plan to devise an approach to deal with this limitation in the future. An important extension of LGR2 is to evaluate and train the framework on linguistically diverse and complex natural language instructions, including ambiguous, multi-step, context-dependent, and

constraint-based commands. This will require enhancing the language-to-reward translator to handle compositional semantics, disambiguate instructions in dynamic contexts, and generate temporally extended subgoal sequences. Additionally, introducing a clarification dialogue module to interactively resolve ambiguous language could further improve robustness in real-world deployments.

Discussion In this work, we propose LGR2, a novel HRL based approach that employs language-guided reward generation to address non-stationarity in HRL. LGR2 effectively leverages natural language instructions to directly solve complex robotic control tasks. We empirically show that LGR2 is able to demonstrate impressive performance on complex robotic control tasks, and is able to significantly outperform the baselines. Additionally, our hierarchical formulation is able to outperform single-level L2R formulation. We therefore believe that LGR2 is a promising step towards building practical robotic systems that use language instructions to solve complex real-world tasks.

Chapter 10

Conclusion

Hierarchical reinforcement learning, first formalized by Dayan and Hinton in 1993 and extensively developed over three decades, represents a fundamental paradigm for decomposing complex sequential decision-making problems into manageable hierarchical structures. While the field has achieved significant theoretical advances and demonstrated success in various domains, two critical limitations have persistently hindered its practical deployment in real-world robotic systems: non-stationarity arising from concurrent training of multiple policy levels and infeasible subgoal generation by higher-level policies.

This thesis specifically addresses these core limitations within the established HRL framework to solve sparse, long horizon tasks by developing novel learning strategies that leverage three complementary signal sources: expert demonstrations, human preferences, and natural language instructions. By integrating advancements in reinforcement learning, imitation learning, preference-based learning, and natural language processing, we propose novel approaches to overcome issues such as non-stationarity, inefficient exploration, and infeasible subgoal generation, which have been the major bottlenecks hindering the scalability of hierarchical reinforcement learning (HRL). The contributions of this work are inspired from neuroscience and cognitive psychology, and involve theoretical frameworks, simulated environments, and real-world robotic implementations. Our work contributes

to the HRL field by providing principled solutions to fundamental challenges that have limited the scalability and reliability of hierarchical approaches in robotic control tasks.

10.1 Specific Contributions to Hierarchical Reinforcement Learning

Addressing Non-stationarity in Multi-level Learning: Building upon foundational work in goal-conditioned HRL, we tackle the well-documented problem of reward and transition function instability that occurs when lower-level primitives evolve during training. Our approaches: CRISP’s curriculum-based parsing, PEAR’s adaptive relabeling, PIPER’s preference-based reward modeling, and HPO’s direct preference optimization provide complementary strategies for decoupling higher-level learning from lower-level policy changes, a challenge that has persisted since early HRL formulations.

Feasible Subgoal Generation: We address the critical issue of infeasible subgoal prediction, which can stall learning at lower levels and degrade overall performance. Through primitive-informed regularization mechanisms that consider the current capabilities of lower-level policies, our methods ensure that hierarchical decomposition remains beneficial rather than detrimental to learning efficiency.

Integration of Modern Learning Paradigms: While classical HRL relied primarily on environment rewards and hand-crafted hierarchies, we demonstrate how contemporary advances in preference-based learning and large language models can be systematically integrated into hierarchical frameworks to improve both sample efficiency and human-interpretability.

10.2 Comparative Summary of Method Contributions:

The methods represent a progression from demonstration-dependent (CRISP, PEAR) to preference-based (PIPER, HPO) to language-guided approaches (LGR2), each reducing

Method	Core Innovation	Supervision Source	Key Technical Contribution	Primary Challenge Addressed	Unique Advantage
CRISP	Curriculum-inducing primitive parsing	Expert demonstrations	Primitive Informed Parsing (PIP) with environment reset capability	Non-stationarity via curriculum generation	Automatic subgoal difficulty progression
PEAR	Q-function guided adaptive relabeling	Expert demonstrations	Lower-level Q-function thresholding for subgoal selection	Non-stationarity without environment reset	Practical deployment without reset assumptions
PIPER	Preference-based hierarchical learning	Human preferences (simulated via PiL)	Primitive-in-the-Loop feedback with hindsight relabeling	Non-stationarity when expert demos unavailable	No expert demonstration requirement
HPO	Bi-level optimization with DPO	Human preferences	Token-level DPO with primitive regularization	Both non-stationarity and infeasible subgoals	Principled bi-level formulation avoiding RL instability
LGR2	Language-guided reward generation	Natural language instructions	LLM-based reward parameter generation	Language-to-action translation in HRL	Direct natural language control interface

TABLE 10.1: Comparative analysis of proposed HRL methods

human annotation requirements while increasing practical applicability and user accessibility.

The extensive experiments across diverse robotic tasks, ranging from navigation to manipulation validate the proposed approaches against state-of-the-art baselines, demonstrating significant improvements in sample efficiency and task completion. Furthermore, the real-world robotic experiments demonstrate the robustness and generalization of the proposed frameworks in practical real-world applications.

10.3 Positioning Within Current HRL Research

Our work addresses a specific gap in the HRL literature: while recent advances have focused on skill discovery and meta-learning approaches, fewer works have systematically tackled the fundamental stability issues that arise in practical hierarchical learning scenarios. Unlike approaches that assume access to optimal lower-level policies or require extensive domain-specific engineering, our methods provide general-purpose solutions applicable across diverse robotic domains.

The experimental validation across six distinct environments, from navigation to manipulation tasks, demonstrates that our approaches scale beyond the simplified domains often

used in HRL research, addressing the real-world deployment challenges that have limited HRL adoption in robotics.

While the proposed hierarchical strategies demonstrate significant improvements in complex robotic control tasks, several important avenues for future research emerge from this work. To motivate future work, we first discuss common failure modes observed across the proposed methods and outlines theoretical directions that warrant deeper investigation.

10.4 Common Failure Cases and Mitigation Strategies

Overfitting to Subgoal Structure. In un-regularized implementations of the proposed algorithms, the higher-level policies show a tendency of overfitting to specific subgoal structures present in expert demonstrations or preference data. This manifests as poor generalization when the optimal subgoal sequence differs significantly from the training distribution. In future work, we would like to investigate better adaptive subgoal regularization techniques that encourage exploration of diverse subgoal structures while maintaining task-relevant constraints. One promising direction involves incorporating uncertainty estimation into subgoal selection, where the higher-level policy actively explores regions of high epistemic uncertainty in the subgoal space.

Brittle Primitive Parsing. The adaptive relabeling mechanisms in CRISP and PEAR, while effective in controlled settings, exhibit sensitivity to hyperparameter choices and environmental stochasticity. The threshold-based parsing in PEAR, for example, can lead to inconsistent subgoal generation when the lower-level value function exhibits high variance during early training phases. To address this brittleness, we could explore robust parsing algorithms that incorporate confidence intervals or ensemble-based approaches for subgoal selection in future work. Additionally, developing self-adaptive threshold mechanisms that adjust based on the current learning progress could improve stability across diverse environments.

Language Misinterpretation in LGR2. The language-guided reward generation in LGR2 occasionally produces reward functions that misalign with the intended task semantics, particularly for ambiguous instructions or novel object-action combinations not seen during language model training. In future work, we could investigate interactive clarification mechanisms where the system can query for additional information when instruction ambiguity is detected. LLMs also suffer from hallucinations, effectively generating sub-optimal reward functions that may lead to poor performance. Although we employ reward translator module to deal with such hallucinations, better prompt descriptions or integrating visual grounding with language processing could help such ambiguities more effectively.

Scalability to Complex Hierarchies. All proposed methods focus on two-level hierarchies, but real-world tasks often benefit from deeper hierarchical structures. Extending these approaches to multi-level hierarchies introduces challenges in credit assignment and coordination across multiple abstraction levels.

10.5 Broader Impact and Future Directions Within HRL

Convergence Analysis. In future work, we could establish convergence conditions for the joint RL-IRL optimization under primitive-informed regularization. Specifically, analyzing the fixed-point properties of the primitive parsing operator and its interaction with policy learning could provide insights into stable convergence regimes.

Sample Complexity of Preference-Based Hierarchical Learning. PIPER’s success in learning from preference data raises fundamental questions about the sample complexity of hierarchical preference learning. The primitive-in-the-loop feedback mechanism introduces dependencies between levels that complicate standard analysis techniques. We could analyze how the quality of lower-level policies affects the sample complexity of higher-level preference learning. Understanding these trade-offs could guide the design of more sample-efficient hierarchical preference algorithms.

Stability Analysis for Multi-Level DPO. HPO’s application of direct preference optimization to hierarchical settings introduces novel theoretical challenges. The bi-level formulation with primitive regularization creates coupled optimization dynamics that warrant formal stability analysis. In future work, we would like to investigate the local stability properties of the proposed bi-level optimization, particularly how perturbations in lower-level policies affect higher-level convergence. Developing sufficient conditions for stable bi-level optimization could extend the applicability of DPO-based methods to more complex hierarchical architectures.

Long-term Vision for HRL in Robotics. The successful real-world deployment demonstrated across multiple robotic platforms indicates that HRL can move beyond proof-of-concept demonstrations toward practical autonomous systems. The language-guided capabilities in LGR2 represent an initial step toward hierarchical systems that can interpret and execute complex instructions, a capability essential for general-purpose robotic assistants.

Human-Robot Interaction. The language-guided capabilities demonstrated in LGR2 represent an initial step toward more natural human-robot interaction. In future, we would like to explore how to extend these capabilities to handle multi-modal instructions, temporal constraints, and collaborative task execution.

Looking ahead, the hierarchical strategies presented in this thesis provide a strong foundation for future research in robotic control. These directions outlined above collectively point toward a future where hierarchical reinforcement learning can provide both theoretical guarantees and practical robustness for complex real-world applications. The foundation established by the methods in this thesis provides a strong starting point for addressing these fundamental challenges in autonomous robotic systems. These directions hold immense potential to advance the goal of creating intelligent, autonomous robotic systems capable of solving real-world problems efficiently and reliably.

While our knowledge of artificial intelligence has expanded tremendously since its inception, I believe we are still standing on the shore of the *great ocean of truth*, and have only

begun to uncover the wonders the world holds. As said by Issac Newton,

*I do not know what I may appear to the world, but to myself, I seem to have been only
like a boy playing on the seashore, and diverting myself in now and then finding a
smoother pebble or a prettier shell than ordinary, whilst the great ocean of truth lay all
undiscovered before me.*

Bibliography

- [1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, “Mastering the game of go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
- [2] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, *et al.*, “Chain-of-thought prompting elicits reasoning in large language models,” *Advances in neural information processing systems*, vol. 35, pp. 24824–24837, 2022.
- [3] J. Jumper, R. Evans, A. Pritzel, *et al.*, “Highly accurate protein structure prediction with alphafold,” *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.
- [4] C. Finn, S. Levine, and P. Abbeel, “Guided cost learning: Deep inverse optimal control via policy optimization,” in *International conference on machine learning*, pp. 49–58, PMLR, 2016.
- [5] M. M. Botvinick, “Hierarchical models of behavior and prefrontal function,” *Trends in cognitive sciences*, vol. 12, no. 5, pp. 201–208, 2008.
- [6] O. Nachum, H. Tang, X. Lu, S. Gu, H. Lee, and S. Levine, “Why does hierarchy (sometimes) work so well in reinforcement learning?,” *arXiv preprint arXiv:1909.10618*, 2019.
- [7] C. Boutilier, T. L. Dean, and S. Hanks, “Decision-theoretic planning: Structural assumptions and computational leverage,” *Journal of Artificial Intelligence Research*, vol. 11, pp. 1–94, 1999.

- [8] C. Boutilier, R. Dearden, and M. Goldszmidt, “Stochastic dynamic programming with factored representations,” *Artificial Intelligence*, vol. 121, no. 1-2, pp. 49–107, 2000.
- [9] D. P. Bertsekas and J. N. Tsitsiklis, “Neuro-dynamic programming,” *Athena Scientific*, 1996.
- [10] C. Boutilier, T. Dean, and S. Hanks, “Decision-theoretic planning: Structural assumptions and computational leverage,” *Journal of Artificial Intelligence Research*, vol. 11, pp. 1–94, 1999.
- [11] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT Press, 1998.
- [12] R. S. Sutton, D. Precup, and S. Singh, “Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning,” *Artificial Intelligence*, vol. 112, no. 1, pp. 181–211, 1999.
- [13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” in *Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS)*, pp. 1–9, Curran Associates Inc., 2013.
- [14] D. J. Felleman and D. C. Van Essen, “Distributed hierarchical processing in the primate cerebral cortex,” *Cerebral cortex (New York, NY: 1991)*, vol. 1, no. 1, pp. 1–47, 1991.
- [15] D. Badre and M. D’esposito, “Is the rostro-caudal axis of the frontal lobe hierarchical?,” *Nature reviews neuroscience*, vol. 10, no. 9, pp. 659–669, 2009.
- [16] K. Currie and A. Tate, “O-plan: The open planning architecture,” in *Knowledge Engineering Review*, vol. 6, pp. 277–291, 1991.
- [17] R. E. Fikes and N. J. Nilsson, “Learning and executing generalized robot plans,” *Artificial Intelligence*, vol. 3, no. 4, pp. 251–288, 1972.

- [18] R. E. Korf, “Macro-operators: A weak method for learning,” in *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1341–1347, 1985.
- [19] C. A. Knoblock, “Automatic abstraction in planning,” in *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI)*, pp. 236–241, 1990.
- [20] E. D. Sacerdoti, “Planning in a hierarchy of abstraction spaces,” Tech. Rep. 68, Artificial Intelligence Center, SRI International, 1974.
- [21] R. S. Sutton, D. Precup, and S. Singh, “Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning,” *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.
- [22] T. G. Dietterich, “Hierarchical reinforcement learning with the MAXQ value function decomposition,” *CoRR*, vol. cs.LG/9905014, 1999.
- [23] P. Dayan and G. E. Hinton, “Feudal reinforcement learning,” in *Advances in Neural Information Processing Systems*, pp. 271–278, Morgan Kaufmann Publishers Inc., 1993.
- [24] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, “Feudal networks for hierarchical reinforcement learning,” in *International conference on machine learning*, pp. 3540–3549, PMLR, 2017.
- [25] P. H. Stone and M. M. Veloso, *Layered learning in multiagent systems*. PhD thesis, Carnegie Mellon University, USA, 1998. AAI9918612.
- [26] O. Nachum, S. Gu, H. Lee, and S. Levine, “Data-efficient hierarchical reinforcement learning,” in *Advances in Neural Information Processing Systems*, vol. 31, pp. 3303–3313, 2018.
- [27] A. Levy, G. Konidaris, R. Platt, and K. Saenko, “Learning multi-level hierarchies with hindsight,” in *International Conference on Learning Representations*, 2018.

- [28] E. Chane-Sane, C. Schmid, and I. Laptev, “Goal-conditioned reinforcement learning with imagined subgoals,” in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, p. TBD, 2021.
- [29] A. L. McGovern and A. G. Barto, “Automatic discovery of subgoals in reinforcement learning using diverse density,” in *ICML*, pp. 361–368, 2001.
- [30] I. Menache, S. Mannor, and N. Shimkin, “Q-cut–dynamic discovery of sub-goals in reinforcement learning,” in *ECML*, pp. 295–306, 2002.
- [31] M. C. Machado, M. G. Bellemare, and M. Bowling, “A laplacian framework for option discovery in reinforcement learning,” in *ICML*, 2017.
- [32] J. Achiam, H. Edwards, D. Amodei, and P. Abbeel, “Variational option discovery algorithms,” in *arXiv preprint arXiv:1807.10299*, 2018.
- [33] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, “Deep reinforcement learning from human preferences,” *Advances in neural information processing systems*, vol. 30, 2017.
- [34] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn, “Direct preference optimization: Your language model is secretly a reward model,” *Advances in Neural Information Processing Systems*, vol. 36, 2024.
- [35] R. Bellman, *Dynamic Programming*. Princeton University Press, 1957.
- [36] R. Bellman, “Dynamic programming,” *Science*, vol. 153, no. 3731, pp. 34–37, 1966.
- [37] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson, 2016.
- [38] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon, “A proposal for the dartmouth summer research project on artificial intelligence,” in *August 31, 1955*, 1956.
- [39] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

- [40] Y. Freund and R. E. Schapire, “A decision-theoretic generalization of on-line learning and an application to boosting,” *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.
- [41] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems*, vol. 25, pp. 1097–1105, 2012.
- [42] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, “High-resolution image synthesis with latent diffusion models,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.
- [43] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” *CoRR*, vol. abs/1706.03762, 2017.
- [44] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, “Language models are few-shot learners,” *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.
- [45] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: A survey,” *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.
- [46] R. S. Sutton, “Learning to predict by the methods of temporal differences,” *Machine Learning*, vol. 3, no. 1, pp. 9–44, 1988.
- [47] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” *Advances in Neural Information Processing Systems*, vol. 12, pp. 1057–1063, 2000.
- [48] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [49] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

- [50] E. D. Sacerdoti, *Planning in a hierarchy of abstraction spaces*. PhD thesis, Stanford University, 1974.
- [51] D. E. Knuth and F. Moore, “Analysis of hierarchical planning systems,” *Artificial Intelligence*, vol. 8, no. 3, pp. 255–288, 1976.
- [52] M. P. Georgeff, “Controlling reasoning: Strategies and heuristics,” in *Proceedings of the 7th international joint conference on Artificial intelligence*, pp. 446–458, 1981.
- [53] L. P. Kaelbling, “Learning to achieve goals,” in *IN PROC. OF IJCAI-93*, pp. 1094–1098, Morgan Kaufmann, 1993.
- [54] R. Parr and S. Russell, “Reinforcement learning with hierarchies of machines,” in *Advances in Neural Information Processing Systems 10*, MIT Press, 1998.
- [55] T. G. Dietterich, “Hierarchical reinforcement learning with the maxq value function decomposition,” *Journal of Artificial Intelligence Research*, vol. 13, pp. 227–303, 2000.
- [56] D. Precup, R. S. Sutton, and S. Singh, “Theoretical results on reinforcement learning with temporally abstract options,” in *European Conference on Machine Learning*, pp. 382–393, Springer, 1998.
- [57] A. G. Barto and S. Mahadevan, “Recent advances in hierarchical reinforcement learning,” *Discrete Event Dynamic Systems*, vol. 13, pp. 341–379, 2003.
- [58] B. Hengst, “Discovering hierarchy in reinforcement learning with hexq,” *ICML*, 2002.
- [59] T. G. Dietterich, “Hierarchical reinforcement learning with the maxq value function decomposition,” in *Journal of Artificial Intelligence Research*, vol. 13, pp. 227–303, 2000.
- [60] A. G. Barto and S. Mahadevan, “Recent advances in hierarchical reinforcement learning,” *Discrete Event Dynamic Systems*, vol. 13, no. 4, pp. 341–379, 2003.
- [61] M. Stolle and D. Precup, “Learning options in reinforcement learning,” in *Abstraction, Reformulation, and Approximation*, pp. 212–223, Springer, 2002.

- [62] S. Mannor, I. Menache, A. Hoze, and U. Klein, “Dynamic abstraction in reinforcement learning via clustering,” in *ICML*, pp. 560–567, 2004.
- [63] M. E. Botvinick, “Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective,” *Cognition*, vol. 113, no. 3, pp. 262–280, 2009.
- [64] P.-L. Bacon, J. Harb, and D. Precup, “The option-critic architecture,” in *AAAI*, pp. 1726–1734, 2017.
- [65] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, “Feudal networks for hierarchical reinforcement learning,” *CoRR*, vol. abs/1703.01161, 2017.
- [66] O. Nachum, S. Gu, H. Lee, and S. Levine, “Data-efficient hierarchical reinforcement learning,” *CoRR*, vol. abs/1805.08296, 2018.
- [67] A. Levy, R. P. Jr., and K. Saenko, “Hierarchical actor-critic,” *CoRR*, vol. abs/1712.00948, 2017.
- [68] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, “Overcoming exploration in reinforcement learning with demonstrations,” in *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 6292–6299, IEEE, 2018.
- [69] K. Shiarlis, M. Wulfmeier, S. Salter, S. Whiteson, and I. Posner, “Taco: Learning task decomposition via temporal alignment for control,” in *Proceedings of the 35th International Conference on Machine Learning*, pp. 4654–4663, PMLR, 2018.
- [70] C. Florensa, Y. Duan, and P. Abbeel, “Stochastic neural networks for hierarchical reinforcement learning,” in *5th International Conference on Learning Representations (ICLR)*, 2017.
- [71] K. Shiarlis, M. Wulfmeier, S. Salter, S. Whiteson, and I. Posner, “TACO: Learning task decomposition via temporal alignment for control,” in *Proceedings of the 35th International Conference on Machine Learning (J. Dy and A. Krause, eds.)*, vol. 80 of *Proceedings of Machine Learning Research*, pp. 4654–4663, PMLR, 10–15 Jul 2018.

- [72] T. Schaul, D. Horgan, K. Gregor, and D. Silver, “Universal value function approximators,” in *Proceedings of the 32nd International Conference on Machine Learning* (F. Bach and D. Blei, eds.), vol. 37 of *Proceedings of Machine Learning Research*, (Lille, France), pp. 1312–1320, PMLR, 2015.
- [73] P. Sharma, B. Sundaralingam, V. Blukis, C. Paxton, T. Hermans, A. Torralba, J. Andreas, and D. Fox, “Correcting robot plans with natural language feedback,” *arXiv preprint arXiv:2204.05186*, 2022.
- [74] J. Lin, D. Fried, D. Klein, and A. Dragan, “Inferring rewards from language in context,” *arXiv preprint arXiv:2204.02515*, 2022.
- [75] M. Kwon, S. M. Xie, K. Bullard, and D. Sadigh, “Reward design with language models,” *arXiv preprint arXiv:2303.00001*, 2023.
- [76] W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. G. Arenas, H.-T. L. Chiang, T. Erez, L. Hasenclever, J. Humplik, *et al.*, “Language to rewards for robotic skill synthesis,” *arXiv preprint arXiv:2306.08647*, 2023.
- [77] T. Howell, N. Gileadi, S. Tunyasuvunakool, K. Zakka, T. Erez, and Y. Tassa, “Predictive sampling: Real-time behaviour synthesis with mujoco,” *arXiv preprint arXiv:2212.00541*, 2022.
- [78] U. Singh and V. P. Namboodiri, “Pear: Primitive enabled adaptive relabeling for boosting hierarchical reinforcement learning,” *arXiv preprint arXiv:2306.06394*, 2023.
- [79] U. Singh, W. A. Suttle, B. M. Sadler, V. P. Namboodiri, and A. S. Bedi, “Piper: Primitive-informed preference-based hierarchical reinforcement learning via hindsight relabeling,” *arXiv preprint arXiv:2404.13423*, 2024.
- [80] U. Singh, S. Chakraborty, W. A. Suttle, B. M. Sadler, A. K. Sahu, M. Shah, V. P. Namboodiri, and A. S. Bedi, “Hierarchical preference optimization: Learning to achieve goals via feasible subgoals prediction,” *arXiv preprint arXiv:2411.00361*, 2024.

- [81] U. Singh, P. Bhattacharyya, and V. P. Namboodiri, “Lgr2: Language guided reward relabeling for accelerating hierarchical reinforcement learning,” *arXiv preprint arXiv:2406.05881*, 2024.
- [82] U. Singh and V. P. Namboodiri, “Crisp: Curriculum inducing primitive informed subgoal prediction,” *arXiv preprint arXiv:2304.03535*, 2023.
- [83] M. A. OpenAI, B. Baker, M. Chociej, R. Jozefowicz, B. Mc-Grew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, *et al.*, “Learning dexterous in-hand manipulation. arxiv e-prints, page,” *arXiv preprint arXiv:1808.00177*, 2018.
- [84] B. Ellenberger, “Pybullet gymperium.” <https://github.com/benelot/pybullet-gym>, 2018–2019.
- [85] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange, “Unity: A general platform for intelligent agents,” *arXiv preprint arXiv:1809.02627*, 2020.
- [86] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, *et al.*, “Multi-goal reinforcement learning: Challenging robotics environments and request for research,” *arXiv preprint arXiv:1802.09464*, 2018.
- [87] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym. arxiv,” *arXiv preprint arXiv:1606.01540*, vol. 10, 2016.
- [88] E. Todorov, T. Erez, and Y. Tassa, “Mujoco: A physics engine for model-based control,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, IEEE, 2012.
- [89] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman, “Relay policy learning: Solving long horizon tasks via imitation and reinforcement learning,” in *Conference on Robot Learning (CoRL)*, 2019.
- [90] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, “D4rl: Datasets for deep data-driven reinforcement learning,” in *Conference on Robot Learning (CoRL)*, pp. 729–735, PMLR, 2020.

- [91] S. Levine, C. Finn, T. Darrell, and P. Abbeel, “End-to-end training of deep visuomotor policies,” *CoRR*, vol. abs/1504.00702, 2015.
- [92] M. Vecerík, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. A. Riedmiller, “Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards,” *CoRR*, vol. abs/1707.08817, 2017.
- [93] A. Rajeswaran, V. Kumar, A. Gupta, J. Schulman, E. Todorov, and S. Levine, “Learning complex dexterous manipulation with deep reinforcement learning and demonstrations,” *CoRR*, vol. abs/1709.10087, 2017.
- [94] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. B. Tenenbaum, “Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation,” *CoRR*, vol. abs/1604.06057, 2016.
- [95] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. H. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, “Hindsight experience replay,” in *NIPS*, 2017.
- [96] P. Dayan and G. E. Hinton, “Feudal reinforcement learning,” in *Advances in Neural Information Processing Systems 5, [NIPS Conference]*, (San Francisco, CA, USA), pp. 271–278, Morgan Kaufmann Publishers Inc., 1993.
- [97] R. Parr and S. Russell, “Reinforcement learning with hierarchies of machines,” in *Advances in Neural Information Processing Systems* (M. Jordan, M. Kearns, and S. Solla, eds.), vol. 10, MIT Press, 1998.
- [98] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, “Overcoming exploration in reinforcement learning with demonstrations,” *CoRR*, vol. abs/1709.10089, 2017.
- [99] T. Hester, M. Vecerík, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, A. Sendonaris, G. Dulac-Arnold, I. Osband, J. P. Agapiou, J. Z. Leibo, and A. Gruslys,

- “Learning from demonstrations for real world reinforcement learning,” *CoRR*, vol. abs/1704.03732, 2017.
- [100] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman, “Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning,” *CoRR*, vol. abs/1910.11956, 2019.
- [101] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, “Curriculum learning,” in *Proceedings of the 26th Annual International Conference on Machine Learning, ICML ’09*, (New York, NY, USA), p. 41–48, Association for Computing Machinery, 2009.
- [102] Y. Song and J. Schneider, “Robust reinforcement learning via genetic curriculum,” in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 5560–5566, IEEE, 2022.
- [103] J. Parker-Holder, M. Jiang, M. Dennis, M. Samvelyan, J. Foerster, E. Grefenstette, and T. Rocktäschel, “Evolving curricula with regret-based environment design,” in *International Conference on Machine Learning*, pp. 17473–17498, PMLR, 2022.
- [104] P. Fournier, O. Sigaud, M. Chetouani, and P.-Y. Oudeyer, “Accuracy-based curriculum learning in deep reinforcement learning,” *arXiv preprint arXiv:1806.09614*, 2018.
- [105] C. Florensa, D. Held, X. Geng, and P. Abbeel, “Automatic goal generation for reinforcement learning agents,” in *International conference on machine learning*, pp. 1515–1528, PMLR, 2018.
- [106] S. Racaniere, A. K. Lampinen, A. Santoro, D. P. Reichert, V. Firoiu, and T. P. Lillicrap, “Automated curricula through setter-solver interactions,” *arXiv preprint arXiv:1909.12892*, 2019.
- [107] Z. Ren, K. Dong, Y. Zhou, Q. Liu, and J. Peng, “Exploration via hindsight goal generation,” *Advances in Neural Information Processing Systems*, vol. 32, 2019.

- [108] A. Sharma, A. Gupta, S. Levine, K. Hausman, and C. Finn, “Autonomous reinforcement learning via subgoal curricula,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 18474–18486, 2021.
- [109] S. K. S. Ghasemipour, R. Zemel, and S. Gu, “A divergence minimization perspective on imitation learning methods,” in *Conference on Robot Learning*, pp. 1259–1277, PMLR, 2020.
- [110] I. Kostrikov, K. K. Agrawal, D. Dwibedi, S. Levine, and J. Tompson, “Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning,” *arXiv preprint arXiv:1809.02925*, 2018.
- [111] J. Ho and S. Ermon, “Generative adversarial imitation learning,” *CoRR*, vol. abs/1606.03476, 2016.
- [112] T. Schaul, D. Horgan, K. Gregor, and D. Silver, “Universal value function approximators,” in *Proceedings of the 32nd International Conference on Machine Learning* (F. Bach and D. Blei, eds.), vol. 37 of *Proceedings of Machine Learning Research*, (Lille, France), pp. 1312–1320, PMLR, 07–09 Jul 2015.
- [113] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, and Z. Wang, “Multi-class generative adversarial networks with the L2 loss function,” *CoRR*, vol. abs/1611.04076, 2016.
- [114] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” *CoRR*, vol. abs/1801.01290, 2018.
- [115] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.
- [116] M. Dalal, D. Pathak, and R. Salakhutdinov, “Accelerating robotic reinforcement learning via parameterized action primitives,” *CoRR*, vol. abs/2110.15360, 2021.
- [117] S. Krishnan, R. Fox, I. Stoica, and K. Goldberg, “Ddco: Discovery of deep continuous options for robot learning from demonstrations,” 2017.

- [118] R. Fox, S. Krishnan, I. Stoica, and K. Goldberg, “Multi-level discovery of deep options,” 2017.
- [119] T. Shankar and A. Gupta, “Learning robot skills with temporal variational inference,” in *ICML*, 2020.
- [120] T. Kipf, Y. Li, H. Dai, V. Zambaldi, A. Sanchez-Gonzalez, E. Grefenstette, P. Kohli, and P. Battaglia, “Compile: Compositional imitation learning and execution,” in *International Conference on Machine Learning*, pp. 3418–3428, PMLR, 2019.
- [121] S. Krishnan, R. Fox, I. Stoica, and K. Goldberg, “DDCO: discovery of deep continuous options for robot learning from demonstrations,” *CoRR*, vol. abs/1710.05421, 2017.
- [122] S. Krishnan, A. Garg, R. Liaw, B. Thananjeyan, L. Miller, F. T. Pokorny, and K. Goldberg, “Swirl: A sequential windowed inverse reinforcement learning algorithm for robot tasks with delayed rewards,” *The International Journal of Robotics Research*, vol. 38, no. 2-3, pp. 126–145, 2019.
- [123] S. Pitis, H. Chan, S. Zhao, B. Stadie, and J. Ba, “Maximum entropy gain exploration for long horizon multi-goal reinforcement learning,” in *International Conference on Machine Learning*, pp. 7750–7761, PMLR, 2020.
- [124] S. Kim, K. Lee, and J. Choi, “Variational curriculum reinforcement learning for unsupervised discovery of skills,” *arXiv preprint arXiv:2310.19424*, 2023.
- [125] B. Eysenbach, S. Gu, J. Ibarz, and S. Levine, “Leave no trace: Learning to reset for safe and autonomous reinforcement learning,” 2017.
- [126] B. Piot, M. Geist, and O. Pietquin, “Boosted bellman residual minimization handling expert demonstrations,” in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD)*, 2014.
- [127] J. Ho and S. Ermon, “Generative adversarial imitation learning,” in *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 4565–4573, 2016.

- [128] X. Mao, Q. Li, H. Xie, R. Y. Lau, and Z. Wang, “Multi-class generative adversarial networks with the l2 loss function,” *arXiv preprint arXiv:1611.04076*, vol. 5, pp. 1057–7149, 2016.
- [129] A. Ajay, A. Kumar, P. Agrawal, S. Levine, and O. Nachum, “Opal: Offline primitive discovery for accelerating offline reinforcement learning,” 2020.
- [130] T. Haarnoja, K. Hartikainen, P. Abbeel, and S. Levine, “Latent space policies for hierarchical reinforcement learning,” in *International Conference on Machine Learning*, pp. 1851–1860, PMLR, 2018.
- [131] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, “Overcoming exploration in reinforcement learning with demonstrations,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6292–6299, IEEE, 2018.
- [132] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pp. 1861–1870, PMLR, 2018.
- [133] M. Dalal, D. Pathak, and R. Salakhutdinov, “Accelerating robotic reinforcement learning via parameterized action primitives,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.
- [134] A. Levy, R. Platt, and K. Saenko, “Hierarchical actor-critic,” in *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018.
- [135] I. Kostrikov, K. K. Agrawal, D. Dwibedi, S. Levine, and J. Tompson, “Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning,” in *7th International Conference on Learning Representations (ICLR)*, 2019.

- [136] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, “Learning complex dexterous manipulation with deep reinforcement learning and demonstrations,” in *Proceedings of Robotics: Science and Systems (RSS)*, Robotics: Science and Systems Foundation, 2018.
- [137] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, A. Sendonaris, G. Dulac-Arnold, I. Osband, J. P. Agapiou, J. Z. Leibo, and A. Gruslys, “Deep q-learning from demonstrations,” in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI)*, pp. 3223–3230, 2018.
- [138] S. Krishnan, R. Fox, I. Stoica, and K. Goldberg, “DDCO: Discovery of deep continuous options for robot learning from demonstrations,” in *Proceedings of the 1st Annual Conference on Robot Learning (CoRL)*, pp. 418–437, 2017.
- [139] S. Krishnan, A. Garg, R. Liaw, B. Thananjeyan, L. Miller, F. T. Pokorny, and K. Goldberg, “Swirl: A sequential windowed inverse reinforcement learning algorithm for robot tasks with delayed rewards,” *The International Journal of Robotics Research*, vol. 38, no. 2-3, pp. 126–145, 2019.
- [140] T. Zhang, S. Guo, T. Tan, X. Hu, and F. Chen, “Generating adjacency-constrained subgoals in hierarchical reinforcement learning,” in *Advances in Neural Information Processing Systems*, vol. 33, pp. 9814–9826, 2020.
- [141] S. Lee, J. Kim, I. Jang, and H. J. Kim, “Dhrl: A graph-based approach for long-horizon and sparse hierarchical reinforcement learning,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, p. TBD, IEEE, 2023.
- [142] A. R. Kreidieh, S. Parajuli, N. Lichtlé, Y. You, R. Nasr, and A. M. Bayen, “Inter-level cooperation in hierarchical reinforcement learning,” in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pp. 2000–2002, 2020.
- [143] K. Pertsch, Y. Lee, and J. J. Lim, “Accelerating reinforcement learning with learned skill priors,” in *Conference on Robot Learning (CoRL)*, pp. 944–957, 2020.

- [144] A. Singh, H. Liu, G. Zhou, T. Yu, P. Abbeel, C. Finn, and S. Levine, “Parrot: Data-driven behavioral priors for reinforcement learning,” in *9th International Conference on Learning Representations (ICLR)*, 2021.
- [145] S. Salter, M. Wulfmeier, D. Tirumala, *et al.*, “Mo2: Model-based offline options,” in *Conference on Lifelong Learning Agents*, pp. 902–919, 2022.
- [146] S. Salter, K. Hartikainen, W. Goodwin, and I. Posner, “Priors, hierarchy, and information asymmetry for skill transfer in reinforcement learning,” in *Proceedings of the 5th Conference on Robot Learning (CoRL)*, p. TBD, PMLR, 2022.
- [147] D. Tirumala, A. Galashov, *et al.*, “Behavior priors for efficient reinforcement learning,” *Journal of Machine Learning Research*, vol. 23, no. 221, pp. 1–68, 2022.
- [148] S. Nasiriany, H. Liu, and Y. Zhu, “Augmenting reinforcement learning with behavior primitives for diverse manipulation tasks,” in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7477–7484, 2022.
- [149] B. Ibarz, J. Leike, T. Pohlen, G. Irving, S. Legg, and D. Amodei, “Reward learning from human preferences and demonstrations in atari,” 2018.
- [150] K. Lee, L. Smith, and P. Abbeel, “Pebble: Feedback-efficient interactive reinforcement learning via relabeling experience and unsupervised pre-training,” 2021.
- [151] A. Y. Ng, D. Harada, and S. Russell, “Policy invariance under reward transformations: Theory and application to reward shaping,” in *Icml*, vol. 99, pp. 278–287, Citeseer, 1999.
- [152] A. Wilson, A. Fern, and P. Tadepalli, “A bayesian approach for policy learning from trajectory preference queries,” in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.
- [153] R. A. Bradley and M. E. Terry, “Rank analysis of incomplete block designs: I. the method of paired comparisons,” *Biometrika*, vol. 39, p. 324, 1952.

- [154] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” in *International Conference on Learning Representations (ICLR)*, 2016.
- [155] S. Levine, “Reinforcement learning and control as probabilistic inference: Tutorial and review,” *arXiv preprint arXiv:1805.00909*, 2018.
- [156] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey, *et al.*, “Maximum entropy inverse reinforcement learning,” in *Aaai*, vol. 8, pp. 1433–1438, Chicago, IL, USA, 2008.
- [157] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *International Conference on Learning Representations (ICLR)*, 2015.
- [158] C. Wirth, R. Akrou, G. Neumann, and J. Fürnkranz, “A survey of preference-based reinforcement learning methods,” *Journal of Machine Learning Research*, vol. 18, no. 136, pp. 1–46, 2017.
- [159] Y. Abdelkareem, S. Shehata, and F. Kararay, “Advances in preference-based reinforcement learning: A review,” *arXiv preprint arXiv:2408.11943*, 2024.
- [160] W. B. Knox and P. Stone, “Interactively shaping agents via human reinforcement: The tamer framework,” in *Proceedings of the fifth international conference on Knowledge capture*, pp. 9–16, 2009.
- [161] W. B. Knox and P. Stone, “Augmenting reinforcement learning with human feedback,” *ICML*, 2011.
- [162] N. A. Vien and W. Ertel, “Learning via human feedback in continuous state and action spaces,” in *AAAI Fall Symposium Series: RL in Real Life (RLIHT)*, pp. 73–77, 2012.
- [163] G. Warnell, N. Waytowich, V. Lawhern, and P. Stone, “Deep tamer: Interactive agent shaping in high-dimensional state spaces,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

- [164] G. Li, H. Hung, S. Whiteson, and W. B. Knox, “Using informative behavior to increase engagement in the tamer framework,” *AAMAS*, 2019.
- [165] A. Wilson, A. Fern, and P. Tadepalli, “A bayesian approach for policy learning from trajectory preference queries,” *Advances in neural information processing systems*, vol. 25, 2012.
- [166] C. Daniel, O. Kroemer, M. Viering, J. Metz, and J. Peters, “Active reward learning with a novel acquisition function,” *Autonomous Robots*, vol. 39, pp. 389–405, 2015.
- [167] P. M. Pilarski, M. R. Dawson, T. Degris, F. Fahimi, J. P. Carey, and R. S. Sutton, “Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning,” in *2011 IEEE international conference on rehabilitation robotics*, pp. 1–7, IEEE, 2011.
- [168] K. Lee, K. Lee, Y. Jeong, S. Lee, J. Shin, and J. Song, “Pebble: Feedback-efficient interactive reinforcement learning via relabeling experience and unsupervised pre-training,” in *International Conference on Machine Learning*, pp. 6111–6120, 2021.
- [169] C. Zhu, A. Toth, S. Sgambelluri, M. Kozachenko, C. G. Northcutt, A. Schwing, and S. Choudhury, “Off-policy reinforcement learning with reward observation,” *arXiv preprint arXiv:2005.03574*, 2020.
- [170] Z. Li, X. Ji, M. Chen, and M. Wang, “Lease: Offline preference-based reinforcement learning with high sample efficiency,” *arXiv preprint arXiv:2301.03076*, 2023.
- [171] Z. Li, X. Ji, M. Chen, and M. Wang, “Sample complexity of preference-based nonparametric off-policy evaluation with deep networks,” *arXiv preprint arXiv:2310.10556*, 2023.
- [172] Z. Li, X. Ji, M. Chen, and M. Wang, “Policy evaluation for reinforcement learning from human feedback: A sample complexity analysis,” in *Proceedings of the 41st International Conference on Machine Learning*, pp. 22992–23034, 2024.

- [173] B. Meng, C. Wirth, H. Wu, M. Huber, and J. Wang, “Seer: Towards efficient preference-based reinforcement learning with active querying and sample-efficient algorithms,” *arXiv preprint arXiv:2209.12322*, 2022.
- [174] Y. Sun, Y. Du, G. Kuo, X. B. Peng, P. Abbeel, and J. Ho, “Reinforcement learning from diverse human preferences,” *arXiv preprint arXiv:2301.13771*, 2023.
- [175] O. Nachum, S. Gu, H. Lee, and S. Levine, “Near-optimal representation learning for hierarchical reinforcement learning,” *arXiv preprint arXiv:1810.01257*, 2018.
- [176] R. Rafailov, J. Hejna, R. Park, and C. Finn, “From r to q^* : Your language model is secretly a q -function,” *arXiv preprint arXiv:2404.12358*, 2024.
- [177] B. Liu, M. Ye, S. Wright, P. Stone, and Q. Liu, “Bome! bilevel optimization made easy: A simple first-order approach,” *Advances in neural information processing systems*, vol. 35, pp. 17248–17262, 2022.
- [178] D. Garg, S. Chakraborty, C. Cundy, J. Song, and S. Ermon, “Iq-learn: Inverse soft-q learning for imitation,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 4028–4039, 2021.
- [179] B. D. Ziebart, *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. Carnegie Mellon University, 2010.
- [180] V. H. Wang, J. Pajarinen, T. Wang, and J.-K. Kämäräinen, “State-conditioned adversarial subgoal generation,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 37, pp. 10184–10191, 2023.
- [181] J. Hejna, R. Rafailov, H. Sikchi, C. Finn, S. Niekum, W. B. Knox, and D. Sadigh, “Contrastive preference learning: Learning from human feedback without rl,” *arXiv preprint arXiv:2310.13639*, 2023.
- [182] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, “Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation,” *CoRR*, vol. abs/1806.10293, 2018.

- [183] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, *et al.*, “Language models are few-shot learners,” *NeurIPS*, vol. 33, pp. 1877–1901, 2020.
- [184] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. Le, D. Zhou, and *et al.*, “Chain of thought prompting elicits reasoning in large language models,” *NeurIPS*, vol. 35, pp. 24824–24837, 2022.
- [185] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, “Large language models are zero-shot reasoners,” in *NeurIPS*, pp. 22199–22213, 2022.
- [186] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, “Code as policies: Language model programs for embodied control,” *arXiv preprint arXiv:2209.07753*, 2023.
- [187] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, *et al.*, “Do as i can, not as i say: Grounding language in robotic affordances,” *arXiv preprint arXiv:2204.01691*, 2022.
- [188] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, *et al.*, “Inner monologue: Embodied reasoning through planning with language models,” *arXiv preprint arXiv:2207.05608*, 2022.
- [189] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman, “Relay policy learning: Solving long horizon tasks via imitation and reinforcement learning,” *Conference on Robot Learning (CoRL)*, 2019.
- [190] K. Pertsch, Y. Lee, and J. J. Lim, “Accelerating reinforcement learning with learned skill priors,” *CoRR*, vol. abs/2010.11944, 2020.
- [191] A. Singh, H. Liu, G. Zhou, A. Yu, N. Rhinehart, and S. Levine, “Parrot: Data-driven behavioral priors for reinforcement learning,” *CoRR*, vol. abs/2011.10024, 2020.
- [192] S. Nasiriany, H. Liu, and Y. Zhu, “Augmenting reinforcement learning with behavior primitives for diverse manipulation tasks,” *CoRR*, vol. abs/2110.03655, 2021.

- [193] H. Kress-Gazit, G. Fainekos, and G. Pappas, “Translating structured english to robot controllers,” *Advanced Robotics*, vol. 22, pp. 1343 – 1359, 2008.
- [194] C. Matuszek, E. V. Herbst, L. Zettlemoyer, and D. Fox, “Learning to parse natural language commands to a robot control system,” in *International Symposium on Experimental Robotics*, 2012.
- [195] A. Ku, P. Anderson, R. Patel, E. Ie, and J. Baldridge, “Room-across-room: Multilingual vision-and-language navigation with dense spatiotemporal grounding,” 2020.
- [196] A. Kamath, P. Anderson, S. Wang, J. Y. Koh, A. Ku, A. Waters, Y. Yang, J. Baldridge, and Z. Parekh, “A new path: Scaling vision-and-language navigation with synthetic instructions and imitation learning,” 2023.
- [197] O. Mees, J. Borja-Diaz, and W. Burgard, “Grounding language with visual affordances over unstructured data,” 2023.
- [198] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn, “Bc-z: Zero-shot task generalization with robotic imitation learning,” 2022.
- [199] C. Lynch, A. Wahid, J. Tompson, T. Ding, J. Betker, R. Baruch, T. Armstrong, and P. Florence, “Interactive language: Talking to robots in real time,” 2022.
- [200] F. Ebert, Y. Yang, K. Schmeckpeper, B. Bucher, G. Georgakis, K. Daniilidis, C. Finn, and S. Levine, “Bridge data: Boosting generalization of robotic skills with cross-domain datasets,” 2021.
- [201] J. Fu, A. K. Balan, S. Levine, and S. Guadarrama, “From language to goals: Inverse reinforcement learning for vision-based instruction following,” *ArXiv*, vol. abs/1902.07742, 2019.
- [202] S. Karamcheti, M. Srivastava, P. Liang, and D. Sadigh, “Lila: Language-informed latent actions,” in *Conference on Robot Learning*, 2021.

- [203] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, “Llama: Open and efficient foundation language models,” 2023.
- [204] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, R. Avila, and I. B. et al., “Gpt-4 technical report,” 2024.
- [205] G. Team, R. Anil, S. Borgeaud, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, and K. M. et al., “Gemini: A family of highly capable multimodal models,” 2024.
- [206] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, *et al.*, “Evaluating large language models trained on code,” *arXiv preprint arXiv:2107.03374*, 2021.
- [207] K. Ellis, C. Wong, M. Nye, M. Sable-Meyer, L. Cary, L. Morales, L. Hewitt, A. Solar-Lezama, and J. B. Tenenbaum, “Dreamcoder: Growing generalizable, interpretable knowledge with wake-sleep bayesian program learning,” 2020.
- [208] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, and C. Sutton, “Program synthesis with large language models,” 2021.
- [209] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago, T. Hubert, P. Choy, C. de Masson d’Autume, I. Babuschkin, X. Chen, P.-S. Huang, J. Welbl, S. Gowal, A. Cherepanov, J. Molloy, D. J. Mankowitz, E. Sutherland Robson, P. Kohli, N. de Freitas, K. Kavukcuoglu, and O. Vinyals, “Competition-level code generation with alphacode,” *Science*, vol. 378, p. 1092–1097, Dec. 2022.
- [210] L. Tian, K. Ellis, M. Kryven, and J. Tenenbaum, “Learning abstract structure for drawing by efficient motor program induction,” in *Advances in Neural Information*

- Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds.), vol. 33, pp. 2686–2697, Curran Associates, Inc., 2020.
- [211] D. Trivedi, J. Zhang, S.-H. Sun, and J. J. Lim, “Learning to synthesize programs as interpretable and generalizable policies,” 2022.
- [212] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, “Code as policies: Language model programs for embodied control,” 2023.
- [213] P. Goyal, S. Niekum, and R. J. Mooney, “Using natural language for reward shaping in reinforcement learning,” 2019.
- [214] S. Nair, E. Mitchell, K. Chen, b. ichter, S. Savarese, and C. Finn, “Learning language-conditioned robot behavior from offline data and crowd-sourced annotation,” in *Proceedings of the 5th Conference on Robot Learning* (A. Faust, D. Hsu, and G. Neumann, eds.), vol. 164 of *Proceedings of Machine Learning Research*, pp. 1303–1315, PMLR, 08–11 Nov 2022.
- [215] D. Bahdanau, F. Hill, J. Leike, E. Hughes, S. Hosseini, P. Kohli, and E. Grefenstette, “Learning to understand goal specifications by modelling reward,” in *International Conference on Learning Representations*, 2018.
- [216] H. Hu and D. Sadigh, “Language instructed reinforcement learning for human-ai coordination,” in *ICML’23*, ICML’23, JMLR.org, 2023.
- [217] J. Lin, D. Fried, D. Klein, and A. Dragan, “Inferring rewards from language in context,” in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (S. Muresan, P. Nakov, and A. Villavicencio, eds.), (Dublin, Ireland), pp. 8546–8560, Association for Computational Linguistics, May 2022.
- [218] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis, “Learning navigation behaviors end-to-end with autorl,” *IEEE Robotics and Automation Letters*, vol. 4, pp. 2007–2014, 2018.

- [219] P. Bacon, J. Harb, and D. Precup, “The option-critic architecture,” *CoRR*, vol. abs/1609.05140, 2016.
- [220] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of Go with deep neural networks and tree search,” *Nature*, vol. 529, pp. 484–489, jan 2016.
- [221] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, “Playing atari with deep reinforcement learning,” *CoRR*, vol. abs/1312.5602, 2013.
- [222] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, “Hindsight experience replay,” in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.
- [223] S. M. Lavalle, “Rapidly-exploring random trees: A new tool for path planning,” tech. rep., ., 1998.
- [224] T. Lillicrap, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.
- [225] M. Laskin, S. Emmons, A. Jain, T. Kurutach, P. Abbeel, and D. Pathak, “Sparse graphical memory for robust planning,” *CoRR*, vol. abs/2003.06417, 2020.
- [226] S. I. Gass and M. C. Fu, eds., *Prim’s Algorithm*, pp. 1160–1160. Boston, MA: Springer US, 2013.
- [227] T. Schaul, D. Horgan, K. Gregor, and D. Silver, “Universal value function approximators,” in *Proceedings of the 32nd International Conference on Machine Learning* (F. Bach and D. Blei, eds.), vol. 37 of *Proceedings of Machine Learning Research*, (Lille, France), pp. 1312–1320, PMLR, 07–09 Jul 2015.

- [228] C. Florensa, Y. Duan, and P. Abbeel, “Stochastic neural networks for hierarchical reinforcement learning,” *CoRR*, vol. abs/1704.03012, 2017.
- [229] M. Klissarov, P. Bacon, J. Harb, and D. Precup, “Learnings options end-to-end for continuous action tasks,” *CoRR*, vol. abs/1712.00004, 2017.
- [230] A. Harutyunyan, P. Vrancx, P. Bacon, D. Precup, and A. Nowé, “Learning with options that terminate off-policy,” *CoRR*, vol. abs/1711.03817, 2017.
- [231] J. Harb, P. Bacon, M. Klissarov, and D. Precup, “When waiting is not an option : Learning options with a deliberation cost,” *CoRR*, vol. abs/1709.04571, 2017.
- [232] A. Harutyunyan, W. Dabney, D. Borsa, N. Heess, R. Munos, and D. Precup, “The termination critic,” *CoRR*, vol. abs/1902.09996, 2019.
- [233] P. Henderson, W. Chang, P. Bacon, D. Meger, J. Pineau, and D. Precup, “Optiongan: Learning joint reward-policy options using generative adversarial inverse reinforcement learning,” *CoRR*, vol. abs/1709.06683, 2017.
- [234] A. R. Kreidieh, S. Parajuli, N. Lichtle, Y. You, R. Nasr, and A. M. Bayen, “Inter-level cooperation in hierarchical reinforcement learning,” *CoRR*, vol. abs/1912.02368, 2019.
- [235] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, V. Kumar, and W. Zaremba, “Multi-goal reinforcement learning: Challenging robotics environments and request for research,” *CoRR*, vol. abs/1802.09464, 2018.
- [236] D. Foster and P. Dayan, “Structure in the space of value functions,” *Machine Learning*, vol. 49, no. 2-3, pp. 325–346, 2002.
- [237] M. Wulfmeier, A. Abdolmaleki, R. Hafner, J. T. Springenberg, M. Neunert, T. Hertweck, T. Lampe, N. Y. Siegel, N. Heess, and M. A. Riedmiller, “Regularized hierarchical policies for compositional transfer in robotics,” *CoRR*, vol. abs/1906.11228, 2019.

- [238] M. Wulfmeier, D. Rao, R. Hafner, T. Lampe, A. Abdolmaleki, T. Hertweck, M. Neunert, D. Tirumala, N. Y. Siegel, N. Heess, and M. A. Riedmiller, “Data-efficient hindsight off-policy option learning,” *CoRR*, vol. abs/2007.15588, 2020.
- [239] G. Zuo, Q. Zhao, J. Lu, and J. Li, “Efficient hindsight reinforcement learning using demonstrations for robotic tasks with sparse rewards,” *International Journal of Advanced Robotic Systems*, vol. 17, no. 1, p. 1729881419898342, 2020.
- [240] Y. Ding, C. Florensa, P. Abbeel, and M. Phielipp, “Goal-conditioned imitation learning,” in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.
- [241] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” in *Computer Science, Engineering*, 1998.
- [242] V. Kumar and e. todorov, “Mujoco haptix: A virtual reality system for hand manipulation,” in *Mujoco*, 11 2015.
- [243] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, “D4RL: datasets for deep data-driven reinforcement learning,” *CoRR*, vol. abs/2004.07219, 2020.
- [244] X. B. Peng, P. Abbeel, S. Levine, and M. Van de Panne, “Deepmimic: Example-guided deep reinforcement learning of physics-based character skills,” *ACM Transactions On Graphics (TOG)*, vol. 37, no. 4, pp. 1–14, 2018.
- [245] T. Salimans and R. Chen, “Learning montezuma’s revenge from a single demonstration,” *arXiv preprint arXiv:1812.03381*, 2018.
- [246] C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel, “Reverse curriculum generation for reinforcement learning,” in *Conference on robot learning*, pp. 482–495, PMLR, 2017.
- [247] S. Levine, C. Finn, T. Darrell, and P. Abbeel, “End-to-end training of deep visuomotor policies,” *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.

- [248] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, *et al.*, “Scalable deep reinforcement learning for vision-based robotic manipulation,” in *Conference on robot learning*, pp. 651–673, PMLR, 2018.
- [249] S. Gu, E. Holly, T. Lillicrap, and S. Levine, “Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3389–3396, IEEE, 2017.
- [250] O. Nachum, H. Tang, X. Lu, S. Gu, H. Lee, and S. Levine, “Why does hierarchy (sometimes) work so well in reinforcement learning?,” in *Advances in Neural Information Processing Systems*, vol. 32, 2019.
- [251] P.-L. Bacon, J. Harb, and D. Precup, “The option-critic architecture,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, 2017.
- [252] M. Klissarov, P.-L. Bacon, J. Harb, and D. Precup, “Learning options end-to-end for continuous action tasks,” in *Proceedings of the 31st Conference on Neural Information Processing Systems (NeurIPS) Hierarchical Reinforcement Learning Workshop*, 2017.
- [253] R. Fox, S. Krishnan, I. Stoica, and K. Goldberg, “Multi-level discovery of deep options,” in *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pp. 1665–1674, PMLR, 2017.
- [254] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, “Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation,” *Advances in neural information processing systems*, vol. 29, 2016.
- [255] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller, “Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards,” in *Proceedings of the 1st Conference on Robot Learning (CoRL)*, pp. 373–382, PMLR, 2017.

- [256] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” in *4th International Conference on Learning Representations (ICLR)*, 2016.
- [257] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” tech. rep., Tech. rep., 1998.
- [258] S. Emmons, A. Jain, M. Laskin, T. Kurutach, P. Abbeel, and D. Pathak, “Sparse graphical memory for robust planning,” in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 12916–12927, Curran Associates, Inc., 2020.
- [259] N. Savinov, A. Dosovitskiy, and V. Koltun, “Semi-parametric topological memory for navigation,” in *International Conference on Learning Representations (ICLR)*, 2018.
- [260] S. I. Gass and M. C. Fu, “Prim’s algorithm,” in *Encyclopedia of Operations Research and Management Science*, p. 1160, Springer US, 2013.
- [261] R. Fox, S. Krishnan, I. Stoica, and K. Goldberg, “Multi-level discovery of deep options,” *arXiv preprint arXiv:1703.08294*, 2017.
- [262] T. Shankar and A. Gupta, “Learning robot skills with temporal variational inference,” in *Proceedings of the 37th International Conference on Machine Learning (ICML)*, 2020.
- [263] B. Eysenbach, S. Gu, J. Ibarz, and S. Levine, “Leave no trace: Learning to reset for safe and autonomous reinforcement learning,” in *6th International Conference on Learning Representations (ICLR)*, 2018.
- [264] S. K. S. Ghasemipour, R. Zemel, and S. Gu, “A divergence minimization perspective on imitation learning methods,” in *Conference on Robot Learning (CoRL)*, pp. 1259–1277, PMLR, 2020.
- [265] T. Schaul, D. Horgan, K. Gregor, and D. Silver, “Universal value function approximators,” in *Proceedings of the 32nd International Conference on Machine Learning*, pp. 1312–1320, PMLR, 2015.

- [266] A. Harutyunyan, P. Vrancx, P.-L. Bacon, D. Precup, and A. Nowé, “Learning with options that terminate off-policy,” in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*, pp. 3201–3208, 2018.
- [267] J. Harb, P.-L. Bacon, M. Klissarov, and D. Precup, “When waiting is not an option: Learning options with a deliberation cost,” in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*, pp. 3206–3213, 2018.
- [268] A. Harutyunyan, W. Dabney, D. Borsa, N. Heess, R. Munos, and D. Precup, “The termination critic,” in *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pp. 2645–2653, 2019.
- [269] P. Henderson, W. D. Chang, P.-L. Bacon, D. Meger, J. Pineau, and D. Precup, “Optiongan: Learning joint reward-policy options using generative adversarial inverse reinforcement learning,” in *Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS)*, pp. 10518–10528, 2018.
- [270] O. Nachum, H. Tang, X. Lu, S. Gu, H. Lee, and S. Levine, “Why does hierarchy (sometimes) work so well in reinforcement learning?,” in *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS)*, 2019.
- [271] A. R. Kreidieh, S. Parajuli, N. Lichtlé, Y. You, R. Nasr, and A. M. Bayen, “Inter-level cooperation in hierarchical reinforcement learning,” in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pp. 2000–2002, 2020.
- [272] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” in *4th International Conference on Learning Representations (ICLR)*, 2016.
- [273] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, “Curriculum learning,” in *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*, pp. 41–48, ACM, 2009.
- [274] L. P. Kaelbling, “Learning to achieve goals,” in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1094–1098, 1993.

- [275] D. Foster and P. Dayan, “Structure in the space of value functions,” *Machine Learning*, vol. 49, no. 2–3, pp. 325–346, 2002.
- [276] M. Wulfmeier, A. Abdolmaleki, R. Hafner, J. T. Springenberg, *et al.*, “Regularized hierarchical policies for compositional transfer in robotics,” *arXiv preprint arXiv:1906.11228*, 2019.
- [277] M. Wulfmeier, D. Rao, R. Hafner, *et al.*, “Data-efficient hindsight off-policy option learning,” in *International Conference on Machine Learning (ICML)*, pp. 11340–11350, 2021.
- [278] G. Zuo, Q. Zhao, J. Lu, and J. Li, “Efficient hindsight reinforcement learning using demonstrations for robotic tasks with sparse rewards,” *International Journal of Advanced Robotic Systems*, vol. 17, no. 1, p. 1729881419898342, 2020.
- [279] Y. Ding, C. Florensa, P. Abbeel, and M. Phielipp, “Goal-conditioned imitation learning,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.
- [280] S. Gu, E. Holly, T. P. Lillicrap, and S. Levine, “Deep reinforcement learning for robotic manipulation,” *CoRR*, vol. abs/1610.00633, 2016.
- [281] P. Dayan and G. E. Hinton, “Feudal reinforcement learning,” *Advances in neural information processing systems*, vol. 5, 1992.
- [282] M. Klissarov, P.-L. Bacon, J. Harb, and D. Precup, “Learnings options end-to-end for continuous action tasks,” *arXiv preprint arXiv:1712.00004*, 2017.
- [283] J. Harb, P.-L. Bacon, M. Klissarov, and D. Precup, “When waiting is not an option: Learning options with a deliberation cost,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.
- [284] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust region policy optimization,” in *International conference on machine learning*, pp. 1889–1897, PMLR, 2015.

- [285] I. Popov, N. Heess, T. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. Riedmiller, “Data-efficient deep reinforcement learning for dexterous manipulation,” *arXiv preprint arXiv:1704.03073*, 2017.
- [286] Z. Cao, K. Wong, and C.-T. Lin, “Human preference scaling with demonstrations for deep reinforcement learning,” *arXiv preprint arXiv:2007.12904*, 2020.
- [287] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, “Hindsight experience replay,” *CoRR*, vol. abs/1707.01495, 2017.
- [288] O. Nachum, H. Tang, X. Lu, S. Gu, H. Lee, and S. Levine, “Why does hierarchy (sometimes) work so well in reinforcement learning?,” *CoRR*, vol. abs/1909.10618, 2019.
- [289] R. Fox, S. Krishnan, I. Stoica, and K. Goldberg, “Multi-level discovery of deep options,” *CoRR*, vol. abs/1703.08294, 2017.
- [290] C. Florensa, Y. Duan, and P. Abbeel, “Stochastic neural networks for hierarchical reinforcement learning,” 2017.
- [291] O. Nachum, S. Gu, H. Lee, and S. Levine, “Near-optimal representation learning for hierarchical reinforcement learning,” 2019.
- [292] N. Savinov, A. Dosovitskiy, and V. Koltun, “Semi-parametric topological memory for navigation,” *CoRR*, vol. abs/1803.00653, 2018.
- [293] B. Piot, M. Geist, and O. Pietquin, “Boosted bellman residual minimization handling expert demonstrations,” in *ECML/PKDD*, 2014.
- [294] T. Zhang, S. Guo, T. Tan, X. Hu, and F. Chen, “Generating adjacency-constrained subgoals in hierarchical reinforcement learning,” *CoRR*, vol. abs/2006.11485, 2020.
- [295] E. Chane-Sane, C. Schmid, and I. Laptev, “Goal-conditioned reinforcement learning with imagined subgoals,” *CoRR*, vol. abs/2107.00541, 2021.

- [296] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, “Large language models are zero-shot reasoners,” *Advances in neural information processing systems*, vol. 35, pp. 22199–22213, 2022.
- [297] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, *et al.*, “Inner monologue: Embodied reasoning through planning with language models,” *arXiv preprint arXiv:2207.05608*, 2022.
- [298] W. B. Knox, S. Hatgis-Kessell, S. Booth, S. Niekum, P. Stone, and A. Alievi, “Models of human preference for learning reward functions,” *arXiv preprint arXiv:2206.02231*, 2022.
- [299] U. Singh, S. Chakraborty, W. A. Suttle, B. M. Sadler, V. P. Namboodiri, and A. S. Bedi, “Dipper: Direct preference optimization to accelerate primitive-enabled hierarchical reinforcement learning,” *arXiv preprint arXiv:2406.10892*, 2024.
- [300] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
- [301] H. Lee, S. Phatale, H. Mansoor, K. Lu, T. Mesnard, C. Bishop, V. Carbune, and A. Rastogi, “Rlaif: Scaling reinforcement learning from human feedback with ai feedback,” *arXiv preprint arXiv:2309.00267*, 2023.
- [302] Y. Duan, J. Schulman, X. Chen, P. Bartlett, I. Sutskever, and P. Abbeel, “ RL^2 : Fast reinforcement learning via slow reinforcement learning,” *arXiv preprint arXiv:1611.02779*, 2016.
- [303] H. Zhang, J. Li, L. Weng, Y. Chen, Y. Kong, and M. Chadwick, “Influencing humans to conform to preference models for rlhf,” *arXiv preprint arXiv:2212.10702*, 2022.
- [304] D. e. a. Kalashnikov, “Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation,” *arXiv preprint arXiv:1806.10293*, 2018.
- [305] S. Gu, E. Holly, T. P. Lillicrap, and S. Levine, “Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates,” *arXiv preprint arXiv:1610.00633*, 2016.

- [306] M. Klissarov, P.-L. Bacon, J. Harb, and D. Precup, “Learnings options end-to-end for continuous action tasks,” *arXiv preprint arXiv:1712.00004*, 2017.
- [307] J. Harb, P.-L. Bacon, M. Klissarov, and D. Precup, “When waiting is not an option: Learning options with a deliberation cost,” in *AAAI Conference on Artificial Intelligence*, vol. 32, 2018.
- [308] U. Singh and V. P. Namboodiri, “Pear: Primitive enabled adaptive relabeling for boosting hierarchical reinforcement learning,” *arXiv preprint arXiv:2306.06394*, 2023.
- [309] U. Singh and V. P. Namboodiri, “Crisp: Curriculum inducing primitive informed subgoal prediction,” *arXiv preprint arXiv:2304.03535*, 2023.
- [310] U. Singh, W. A. Suttle, B. M. Sadler, V. P. Namboodiri, and A. Bedi, “Piper: Primitive-informed preference-based hierarchical reinforcement learning via hind-sight relabeling,” *arXiv preprint arXiv:2404.13423*, 2024.
- [311] U. Singh, S. Chakraborty, W. A. Suttle, B. M. Sadler, A. K. Sahu, M. Shah, V. P. Namboodiri, and A. S. Bedi, “Hierarchical preference optimization: Learning to achieve goals via feasible subgoals prediction,” *arXiv preprint arXiv:2411.00361*, 2024.
- [312] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, *et al.*, “Evaluating large language models trained on code,” *arXiv preprint arXiv:2107.03374*, 2021.
- [313] P. Goyal, S. Niekum, and R. J. Mooney, “Using natural language for reward shaping in reinforcement learning,” *arXiv preprint arXiv:1903.02020*, 2019.
- [314] J. Lin, D. Fried, D. Klein, and A. Dragan, “Inferring rewards from language in context,” *arXiv preprint arXiv:2204.02515*, 2022.
- [315] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, “Hindsight experience replay,” in *NeurIPS*, 2017.

- [316] D. Bahdanau, F. Hill, J. Leike, E. Hughes, S. G. Hosseini, P. Kohli, and E. Grefenstette, “Learning to understand goal specifications by modelling reward,” *International Conference on Learning Representations*, 2018.
- [317] S. Nair, E. Mitchell, K. F. Chen, B. Ichter, S. Savarese, and C. Finn, “Learning language-conditioned robot behavior from offline data and crowd-sourced annotation,” *CoRL, PMLR*, vol. 164, pp. 1303–1315, 2022.
- [318] H. Hu and D. Sadigh, “Language instructed reinforcement learning for human-ai coordination,” *Proceedings of the 40th International Conference on Machine Learning*, 2023.
- [319] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, “Llama: Open and efficient foundation language models,” *arXiv preprint arXiv:2302.13971*, 2023.
- [320] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, *et al.*, “Gpt-4 technical report,” *arXiv preprint arXiv:2303.08774*, 2024.
- [321] G. Team, R. Anil, S. Borgeaud, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. Dai, A. Hauth, *et al.*, “Gemini: A family of highly capable multimodal models,” *arXiv preprint arXiv:2312.11805*, 2024.
- [322] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, *et al.*, “Competition-level code generation with alphacode,” *Science*, vol. 378, no. 6624, pp. 1092–1097, 2022.
- [323] L. Tian, K. Ellis, M. Kryven, and J. Tenenbaum, “Learning abstract structure for drawing by efficient motor program induction,” in *NeurIPS*, pp. 2686–2697, 2020.
- [324] D. Trivedi, J. Zhang, S. Sun, and J. J. Lim, “Learning to synthesize programs as interpretable and generalizable policies,” *arXiv preprint arXiv:2108.13643*, 2022.

-
- [325] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *International Conference on Machine Learning (ICML)*, 2018.