Policy Distillation in Offline Multi-Task Reinforcement Learning

by

J.A.E. van Lith

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on November 4 at 10:30AM.

Student number:	4917634	
Project duration:	February 8, 2024 – November 4, 2024	
Thesis committee:	Prof. dr. M.T.J. Spaan,	TU Delft, Chair
	MSc. D. Mambelli,	TU Delft, Supervisor
	Dr. sc. N.M. Gürel,	TU Delft, External examiner

An electronic version of this thesis is available at repository.tudelft.nl.
The official code repository for this project is available at github.com/jaevanlith/mop.

TU Delft

Abstract

In Reinforcement Learning (RL), an agent learns to make decisions by interacting with an environment and receiving feedback in the form of rewards. Multi-Task Reinforcement Learning (MTRL) extends this concept by training a single agent to perform multiple tasks simultaneously, allowing for more efficient use of resources and behavior sharing between tasks. Policy Distillation (PD) is a technique commonly used in MTRL, where policies from multiple single-task agents (teachers) are distilled into a single multi-task agent (student). This is done by merging common structure across tasks, while separating task-specific properties. However, existing PD approaches require interactions with the environment during training.

In this work, we investigate the effectiveness of PD in the offline setting, where the agent has no interaction with the environment before deployment and can only learn from previously collected data. Through a series of experiments, we demonstrate that a straightforward approach yields the highest performance. This approach involves first learning teacher policies using an existing offline RL algorithm, then distilling these policies into a student by sampling states from the offline data and applying a Mean Squared Error (MSE) loss between the teachers' and student's best actions. Moreover, we investigate the effect of a state distribution shift—a major challenge in offline RL—on our approach. We find that such shifts impact performance only slightly in cases of relatively small neural networks or substantial distribution shifts.

We also explore how PD can be enhanced to better capture common structure across related tasks, a key to improving efficiency in MTRL. To this end, we formally define common structure at two levels: the trajectory level and the computational level. To the best of our knowledge, we present the first attempt to quantify the amount of common structure shared across tasks. This measurement reveals that task commonalities are not fully exploited automatically. At the computational level, we attempt to improve sharing of common structure by reducing the network size and adding a regularization term to the loss function. To capture more common structure at the trajectory level, we argue that multi-task exploration is required, meaning that behaviors from one task must be evaluated in the context of another task. We propose two extensions to our approach that introduce multi-task exploration: Data Sharing (DS) and Offline Q-Switch (OQS). While these extensions are capable of improving performance, they also have clear limitations.

Overall, we propose a new, high-performing offline MTRL method and provide valuable insights into the fundamental capabilities and limitations of PD in capturing common structure across tasks, specifically within the offline MTRL setting.

Contents

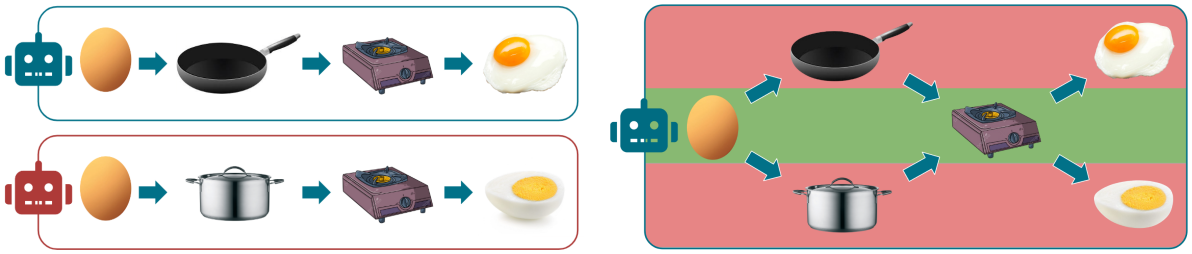
1

Introduction

The field of Reinforcement Learning (RL) has shown notable successes across many challenging domains. RL agents have excelled in complex games [7, 63], continuous control tasks [15, 21], and have even been applied in natural language processing tasks [71, 41]. However, many of these successes were obtained through single-task learning. This means that each task requires a different agent, which is inefficient when tasks are highly related [26]. For instance, consider two related tasks: frying an egg and boiling an egg. Frying an egg roughly involves four phases: grasping an egg, grasping a frying pan, placing it on the stove, and waiting for the egg to fry. For simplicity, details like cracking the egg and pouring oil are omitted. Similarly, boiling an egg involves grasping an egg, grasping a boiling pan, placing it on the stove, and waiting for the egg to boil. This example is visualized in Figure 1.1a. These tasks are highly related, as they require similar skills. Thus, it is inefficient to maintain a separate agent to perform each task.

Similar to frying and boiling eggs, there are many real-world domains in which tasks are related. For example, in robotic manipulation, an agent should be able to manipulate (grasp, stack, push, etc.) different types of objects [31]. While the objects can differ significantly in shape and size, it would be highly inefficient to maintain a separate robot arm for each type of object. Another example is autonomous driving, which includes tasks such as lane following and intersection crossing [13]. These tasks are also highly related, as they both require steering and speed control. It would be inefficient to switch agents when crossing an intersection or entering the highway. Ideally, a single agent should be capable of handling all driving scenarios, just like humans.

There are several RL settings in which agents are trained on various tasks, such as Transfer Learning, Meta RL, and Multi-Task RL (MTRL). The goal is to exploit structural similarities between tasks to enhance generalization [45]. Transfer Learning in RL focuses on transferring knowledge learned from one or more tasks to another related task [60]. The agent first learns a policy on one or more source tasks, then extracts and transfers the reusable knowledge to perform better on a single target task. Meta



(a) Single-task agents: the top agent fries an egg, and the bottom agent boils an egg.

(b) Multi-task agent: common structure is in green, and task-specific properties are in red.

Figure 1.1: Frying and boiling an egg can be done by two single-task agents or one multi-task agent.

RL, on the other hand, teaches the agent to 'learn how to learn' [6]. Instead of transferring knowledge to a particular target task, the agent is trained across a set of tasks to quickly adapt to any new related task. MTRL involves training an agent on multiple tasks simultaneously [64]. The agent attempts to leverage commonalities across tasks to perform well on all of them. Before, we addressed the inefficiency to maintain separate agents for related tasks. Therefore, we focus on MTRL, which aims to learn a single agent capable of performing well across a set of tasks.

To be successful in MTRL, it is essential to capture the common structure between tasks while distinguishing task-specific properties [14]. This approach is particularly useful when tasks are highly related, as in the example of frying and boiling an egg. MTRL aims to learn which skills are similar and can be reused across tasks, such as grasping an egg and putting a pan on the stove. However, the challenge lies in avoiding *negative transfer*, where task-specific properties are inappropriately shared across tasks [40]. For instance, using a boiling pan when frying an egg will lead to a poorly fried egg. Therefore, it is crucial to ensure that the type of pan is not transferred from one task to the other. Figure 1.1b visualizes how a multi-task agent merges common structure in green and distinguishes task-specific properties in red. Note that this example is highly simplified. In reality, common structure and task-specific properties can be intertwined. For example, placing a frying pan or a boiling pan on the stove is similar in terms of keeping the pan upright, but a frying pan may require one hand, while a boiling pan may need two hands. This intertwined nature already indicates the complexity of the problem.

There are several objectives for using MTRL. It can enhance the average performance across tasks [61], compress multiple agents into a smaller agent [51], reduce the overall training time [77], or improve sample efficiency across tasks [10].

A well-studied technique for MTRL focusing on compression while improving performance, is Policy Distillation (PD) [51]. PD was originally designed to compress the size of a single-task policy network (a Deep Neural Network [28]) while maintaining or even improving accuracy. This is achieved by transferring knowledge from a larger, pre-trained network (the teacher) to a smaller network (the student). There are several reasons why this is beneficial. First, training the small network from scratch often yields lower accuracy, because it lacks the capacity to learn as effectively without the guidance of the larger model [27]. During training, a large network can memorize many state-action pairs and quickly adapt to them. This flexibility is necessary, as it is unclear which pairs are most important for achieving high performance. After training, the most relevant state-action pairs become clear, allowing the network to focus on these specifically. Through PD, only the essential knowledge is transferred to the smaller student network. Secondly, deploying the large network, on the other hand, is more computationally expensive [27]. Lastly, the distillation process itself can have a regularizing effect, which sometimes leads to the student outperforming its teacher [51].

In the multi-task setting, there are teachers for each task, and their knowledge is distilled into a single student. This student is designed to solve all tasks while having a size comparable to a single-task teacher. Training the multi-task student from scratch is prone to negative transfer [61], making PD a more effective approach.

Numerous methods have been proposed for multi-task PD and have shown effectiveness in online MTRL. The paper that introduced PD [51] experiments with different distillation loss functions and demonstrates that a student can outperform its teacher by leveraging the Kullback-Leibler divergence. Actor-Mimic [48] achieved similar results using a cross-entropy loss. Distral [61] is another successful method, which introduces a form of multi-task exploration by allowing the student to regularize its teachers. DROID [29] leverages PD to learn autonomous driving across a range of challenging domains. KTM-DRL [70] is the first method to explore the capabilities of PD in the continuous control setting and also shows that the student can outperform its teachers in certain situations.

However, all these methods require online interactions. In many real-world domains, such as healthcare and robotics, online policy exploration is expensive or even dangerous [17]. Offline RL addresses this by training an agent without interaction with the environment, relying instead on previously collected data from a so-called behavior policy [38]. The goal is to safely improve upon this behavior policy, avoiding catastrophic behavior in the environment. An intuitive analogy is learning to fry or boil eggs only by watching video tutorials before attempting the tasks, thereby avoiding the risk of burning down

your house through trial and error. A major challenge in offline RL is the *distribution shift* between the previously collected data and the learned policy [69]. This occurs when the learned policy is significantly different from the behavior policy and can degrade performance upon deployment. To mitigate the distribution shift, the learned policy can be constrained to stay close to the behavior policy [37] or to stay within the known state-action distribution [5].

The combination of the offline and MTRL settings is a relatively under explored research area. Some methods, such as Pre-Training for Robots [36], MT-Opt [32], and Q-Transformer [12], focus on learning a multi-task policy by utilizing large amounts of offline data in high-capacity models. Skills Regularized Task Decomposition (SRTD) [74] is another approach which aims to learn task and skill embeddings to uncover commonalities across tasks. However, none of these methods leverage PD, despite its demonstrated effectiveness in the online setting.

This study investigates the effectiveness of PD in offline MTRL. Additionally, we explore how PD can better capture common structure across related tasks. The combination of multi-tasking and offline learning addresses two important aspects of open problems in RL [45]: generalization (solving a set of tasks simultaneously) and deployability (avoiding catastrophic behavior in the environment). Specifically, this research addresses the following questions:

1. How does Policy Distillation operate effectively in the Offline Multi-Task Reinforcement Learning setting?
2. How can the common structure across related tasks be better captured through Policy Distillation?

The first question focuses on the differences between the online and offline setting relevant to PD. The second question examines PD from a more fundamental multi-tasking perspective.

1.1. Contributions

We will establish a general approach and explore different instances of it, to operate PD effectively in the offline setting. Experiments indicate that a simple method, which performs distillation using a Mean Squared Error (MSE) loss and samples states from the offline data, performs particularly well. Moreover, we investigate the effect of a state distribution shift between the teacher policies and the corresponding offline data on the student's performance. Experiments show that this only becomes problematic when the student has a relatively small size or when the distribution shift is very large. We also find that slightly more diversity in the offline data can outperform the online setting for students with limited network capacity. Moreover, we hypothesize that large distribution shifts can be addressed by increasing the network size. However, results indicate that this approach is prone to overfitting to the offline data.

Additionally, we make several attempts to better capture common structure. First, we formally define common structure at two levels: the trajectory level and the computational level. Next, we design a novel method to measure shared computation, which serves as an indicator of the extent of common structure captured. To the best of our knowledge, this is the first attempt to quantify the amount of common structure shared across tasks. In our initial approach, a significant amount of shared computation is already present for well-performing teachers. However, suboptimal teachers share considerably less computation across tasks.

To improve the sharing of common structure at the computational level, we reduce the network size and introduce a regularization term in the loss function. Reducing the network size is intended to act as an information bottleneck, encouraging the student to merge common behaviors. Nonetheless, this approach does not enhance the sharing of common structure, as it lacks an explicit incentive. To address this, we add a regularization term in the loss function to provide such an incentive explicitly.

We then shift focus to common structure at the trajectory level. We argue that multi-task exploration is essential for capturing this effectively, which requires access to the reward functions. Assuming the reward functions are known, we explore two approaches: Data Sharing (DS) and Offline Q-Switch (OQS). These approaches yield some performance gains but also reveal clear limitations. Overall, our work provides insights into the limitations of PD in capturing common structure within the offline setting.

1.2. Outline

This report is structured as follows:

- Chapter 2 provides the background. It explains the required prior knowledge and introduces the notations used throughout the work.
- Chapter 3 reviews the related work, including online multi-task PD methods, offline MTRL algorithms, and data sharing approaches.
- Chapter 4 details our approach to offline multi-task PD. We discuss two main online approaches and explain the design choices made to extend these to the offline setting. The chapter also highlights the differences between the online and offline approaches. This is followed by a description of the experimental setup, including the environment, tasks, and collection of the offline datasets.
- Chapter 5 addresses the first research question, focusing on applying PD in the offline RL setting. We establish a standard setup by selecting an offline RL algorithm to learn teacher policies, demonstrating that direct multi-tasking suffers from negative transfer, and comparing the performance of three different distillation loss functions. We then examine the effect of a state distribution shift on our approach. Based on the results from the standard setup, we establish a hypothesis and attempt to verify it through a series of experiments.
- Chapter 6 explores the second research question, aimed at better capturing common structure through PD. The chapter begins with a formal definition of common structure, as far as this is feasible. Then, we design a method to measure shared computation across tasks, which indicates how much common structure is captured. We first attempt to increase shared computation through ranking regularization. Next, we assume the reward function is known and implement two approaches that facilitate multi-task exploration: Data Sharing (DS) and Offline Q-Switch (OQS). The chapter concludes with a discussion on the limitations of PD in the offline setting.
- Chapter 7 provides a general discussion on the findings of this work and broader related topics.
- Chapter 8 concludes the report with a brief summary and a recap of the main findings.

2

Background

In this chapter, we introduce the key concepts needed to understand this work. We start by explaining the Markov Decision Process (MDP), which is the basis for decision-making in RL. We then extend this to the multi-task setting and discuss how RL agents learn to perform multiple tasks. After that, we cover RL concepts, including Q-value functions, deep RL, and the offline RL setting, where the agent learns from previously collected data. We also describe the main RL algorithms used in our experiments. Finally, we explain PD. These concepts will be important for understanding the methods and experiments in the rest of the report.

2.1. Markov Decision Process

A Markov Decision Process (MDP) [49] is a mathematical framework used to model decision-making where outcomes are influenced by both the actions taken by an agent and the randomness in the environment. Formally, an MDP is defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \rho, P, R, \gamma)$, where:

- $\mathcal{S}$ is the state space.
- $\mathcal{A}$ is the action space.
- $\rho : \Delta(\mathcal{S})$ is the probability distribution over starting states.
- $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the stochastic transition probability function.
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the deterministic reward function.
- $\gamma \in [0, 1)$ is the discount factor, which determines the importance of future rewards.

An agent operating in an MDP $\mathcal{M}$ will start a state drawn from the distribution over start states: $s_0 \sim \rho$. At each discrete time-step t, it will observe state s_t and take action a_t. The agent will then transition to a next state determined by the transition dynamics: $s_{t+1} \sim P(s_t, a_t)$. It will receive a reward determined by the reward function: $r_t = R(s_t, a_t, s_{t+1})$.

We define a policy π as a function mapping a state to an action, which can be either deterministic $\pi : \mathcal{S} \to \mathcal{A}$ or stochastic $\pi : \mathcal{S} \to \Delta(\mathcal{A})$. The objective in an MDP is to find an optimal policy π^*, which maximizes the expected discounted cumulative reward, also known as the return. The optimal policy is defined as:

$$\pi^*(s) := \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{h-1} \gamma^t R(s_t, a_t, s_{t+1}) \right]$$

$$:= \arg\max_{\pi} \mathbb{E} \left[\sum_{t=0}^{h-1} \gamma^t R(s_t, a_t, s_{t+1}) | s_0 \sim \rho, s_{t+1} \sim P(s_t, a_t), a_t \sim \pi(s_t) \right]$$

where h is the episode length (or horizon), t is the time step, and π is any policy being evaluated.

2.1.1. Multi-Task Markov Decision Process

A multi-task MDP [4] is defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \rho, P, \{R_i, i\}_{i=0}^{m-1}, \gamma)$, which has m tasks with their own index i and reward function R_i. The reward function differs across tasks, whereas the transition function remains the same. This makes it a subset of a Contextual MDP [22], where both the reward and transition function can vary depending on the context.

A multi-task policy $\pi : \mathcal{S} \times i \to \mathcal{A}$ is defined as function mapping a state and task ID to an action. It is aimed to maximize the discounted return across all tasks on average. The optimal multi-task policy is defined as:

$$\pi^*(s, i) := \arg\max_{\pi} \frac{1}{m} \sum_{i=0}^{m-1} \mathbb{E}_{\pi(\cdot|\cdot,i)} \left[\sum_{t=0}^{h-1} \gamma^t R_i(s_t, a_t, s_{t+1}) \right]$$

where h is the episode length (or horizon), t is the time step, and π is any policy being evaluated.

2.2. Reinforcement Learning

In Reinforcement Learning (RL) [30] an agent learns to make decisions by interacting with its environment, formulated as an MDP $\mathcal{M}$. The agent's goal is to learn the optimal policy π^* in its environment. RL methods can broadly be categorized into two main approaches [3]: policy-based and value-based.

In policy-based methods, the agent directly learns the optimal policy by optimizing the expected return of the agent's actions [68]. In contrast, value-based methods involve learning value functions, which estimate the expected return of states or state-action pairs, and use these estimates to derive a policy [66]. In addition, there are approaches that combine both techniques [20].

In the following, we will describe Q-value functions, which are commonly used in value-based approaches. Then, we will introduce deep RL, which employs Deep Neural Networks (DNNs) [28] for value function or policy approximation. Next, we will discuss offline RL, the setting where an agent cannot interact with the environment before deployment. Lastly, we will explain offline Multi-Task Reinforcement Learning (MTRL), the specific setting investigated in this work.

2.2.1. Q-value Function

One of the most common value functions in RL is the Q-value function [57]. The Q-value function $Q^\pi(s, a)$ represents the expected return of taking action a in state s and following policy π thereafter. Formally, the Q-value function is defined as:

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi \left[r_t + \gamma Q^\pi(s_{t+1}, a_{t+1}) \right]$$

where a_{t+1} represents the next action drawn from policy π, s_{t+1} is the next state arrived at after taking action a_t in s_t, and r_t is the reward obtained by taking action a_t in state s_t and ending up in s_{t+1}.

The goal in value-based RL is to learn the optimal Q-value function $Q^* = Q^{\pi^*}$, which gives the highest expected return for any state-action pair. This is defined as:

$$Q^*(s_t, a_t) := \mathbb{E}_{\pi^*} \left[r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \right]$$

Estimating a Q-value function in large, continuous state and action spaces can be done using DNNs. Deep RL [3] is a specific field within RL that employs DNNs as function approximators, which will be explained next.

2.2.2. Deep Reinforcement Learning

In deep RL, Deep Neural Networks (DNNs, or simply referred to as networks) [28] are used to approximate value functions or policies [3]. Instead of defining the state-action mappings in a tabular fashion, DNNs are trained to generalize over large state and action spaces. This makes deep RL effective in complex environments. Deep Q-Networks (DQNs) [44], for example, use DNNs to approximate the

Q-value function. In this work, we leverage DNNs for approximation of both Q-value functions and policies.

2.2.3. Offline Reinforcement Learning

Offline RL is a setting where the agent cannot interact with the environment during training and learns only from previously collected data [38]. This approach helps avoid catastrophic behavior in the environment, making it particularly useful in domains where policy exploration is expensive or dangerous, such as robotics and healthcare [17]. The agent has access to an offline dataset, which consists of transitions (s, a, r, s') previously collected by a behavior policy π_β. The objective of Offline RL is to train a target policy π_{target}, which performs well when deployed in the environment, without the need for collecting any additional data.

However, in Offline RL, the transitions in the dataset are generated by the behavior policy π_β, and the agent must estimate $Q^{\pi_{\text{target}}}(s, a)$ based only on this offline data. This poses a challenge because the state-action distribution induced by π_β may differ significantly from that of π_{target}. This distribution shift [69] occurs when the distribution of states in the offline dataset differs from the distribution of states the agent will encounter during deployment. Since the agent cannot interact with the environment to gather new data, it must generalize from the limited offline data, and any mismatch between the training and deployment distributions can significantly degrade performance.

Offline RL algorithms must address this distribution shift by either constraining the learned policy to stay close to the behavior policy [37] or by penalizing uncertain state-action pairs [5].

2.2.4. Offline Multi-Task Reinforcement Learning

We focus on offline MTRL [76], which is similar to offline RL in many aspects. However, it differs in that it should solve a multi-task MDP and it has access to a multi-task offline dataset containing transitions previously collected by *all* task-specific behavior policies π_{β_i}. The multi-task offline dataset is defined as $\mathcal{D} = \cup_{i=0}^{m-1} \mathcal{D}_i$, where $\mathcal{D}_i = (s, a, r_i, s')$ is the offline dataset for task i, collected by its respective behavior policy π_{β_i}, with rewards determined by R_i.

2.3. Reinforcement Learning Algorithms

In this section, we will describe RL algorithms relevant to our research. This includes actor-critic algorithms, offline algorithms, and PD.

2.3.1. Actor-Critic Reinforcement Learning Algorithms

Actor-critic algorithms are aimed at learning an optimal policy by combining elements of value-based and policy-based approaches [20]. These methods involve two main components: an actor and a critic. The actor represents the current policy π, mapping a state to an action. The critic maintains a Q-value function Q^π, which estimates the value of the actions chosen by the actor. During training, the critic guides the actor by evaluating the taken actions. This evaluation is used to update both the actor (improving the policy) and the critic (improving the Q-value function). The process of alternately improving the policy and updating the value estimates is known as the policy iteration cycle [55].

We will use the actor-critic algorithm Twin Delayed Deep Deterministic Policy Gradient (TD3) [19]. TD3 is an improved version of the Deep Deterministic Policy Gradient (DDPG) [39] algorithm, which was specifically designed for continuous action spaces. TD3 uses two critics to estimate the Q-values and takes the minimum of the two estimates to reduce overestimation bias. The critic is updated by minimizing the Temporal Difference (TD) error [55], while the actor is updated using policy gradients by maximizing the critic. Furthermore, TD3 updates the actor less frequently than the critic to ensure more stable training. The authors recommend one policy update for every two Q-value function updates. Moreover, TD3 performs regularization by adding clipped noise to the target action. This reduces the likelihood of overfitting to a specific action.

2.3.2. Offline Reinforcement Learning Algorithms

As mentioned before, offline RL agents should learn a policy from previously collected data and cannot interact with the environment before deployment. A major challenge is the distribution shift between

the behavior and target policy [17]. To mitigate this, offline RL algorithms must prevent the target policy to visit state-action pairs not present in the offline data.

One of the first methods proposed for this purpose is Conservative Q-learning (CQL) [37]. CQL attempts to mitigate the distribution shift by constraining the target policy to stay close the behavior policy with the use of the Kullback-Leibler (KL) divergence. The disadvantage of this approach is that the performance is highly reliant on the quality of the behavior policy. Alternatively, uncertainty-based approaches can be used which pessimistically set the value of state-action pairs underrepresented in the offline data.

In this work, we will use the uncertainty-based algorithm Pessimistic Bootstrapping for offline RL (PBRL) [5]. PBRL employs an ensemble of critics to quantify uncertainty. This means that the critic consists of multiple Q-value functions, which will output similar values on state-action pairs within the known distribution. However, the Q-value functions will disagree outside of the known distribution. Therefore, the ensemble can be used as an uncertainty quantifier. PBRL penalizes uncertain state-action pairs by taking the minimum value across the Q-value functions. The actor, in turn, maximizes the minimum value across the ensemble of critics. This approach encourages focusing on state-action pairs that are well-represented in the offline data. This approach discourages the agent to visit uncertain states, thereby avoiding catastrophic behavior upon deployment in the environment.

2.3.3. Policy Distillation

Policy Distillation (PD) [51] is a technique designed to compress a large policy network into a smaller network while maintaining or improving performance. It uses a pre-trained network, referred to as the teacher, to train a smaller student network by mimicking the teacher's outputs. A state is sampled from the environment, and the teacher policy computes the action it would take. The student, which is randomly initialized, also predicts an action for that state. The student learns by minimizing a loss function that measures the difference between its outputs and those of the teacher.

The distillation process is a form of Supervised Learning (SL) [24]. In SL, a model is trained to predict an output (the label) based on an input feature vector. The goal is to accurately predict the label for unseen data. The training data consists of pairs of feature vectors and their corresponding ground-truth labels. The model learns by minimizing the difference between its predictions and these ground-truth labels. In the context of PD, the input feature vector corresponds to the input state, the ground-truth label is the teacher's output, and the student is the model attempting to predict the labels. By learning from the teacher's outputs in this supervised fashion, the student approximates the teacher's policy. When the student is deployed in the environment, it must select the same action as its teacher even for unseen states.

PD benefits from the advantages of a large network during training and a small network upon deployment [27]. During training, a relatively large network and extensive training are needed to achieve high performance [51]. Specifically for DQNs in Atari games, it has been shown that training a smaller network results in considerably lower performance [43]. This is because a smaller network has limited capacity to capture complex relationships between states and actions. The authors of the original PD paper [51] speculate that, with fewer parameters, the network can struggle to learn the most important patterns, especially when high-reward situations do not occur often. As a result, it adjusts slower to new and important information, making it harder to find better policies. On the other hand, larger networks can store more information and update faster when relevant state-action pairs are encountered. Therefore, while a smaller network is more efficient for deployment, a larger network is capable of achieving better performance during training. PD trains a relatively large network but distills relevant knowledge to a smaller network for deployment, thereby exploiting the advantages of both.

Additionally, the distillation process can have a regularizing effect, which can lead to the student outperforming its teacher [51]. Since the teacher network is larger, it can approximate more complex functions. This might result in overfitting to states encountered while training. The smaller student network replicates the teacher's behaviors but has fewer parameters, and therefore approximates simpler relationships between the same states and actions. This can lead to better generalization, particularly in continuous state spaces where it is impossible to visit all states during training, increasing the likelihood of overfitting [79].

PD was initially designed for the single-task setting. However, PD can also be effectively used in multi-tasking. Multi-task PD merges knowledge from multiple teacher networks, each trained on a specific task, into a single student network having the same size as each of the teachers. Depending on the specific algorithm, either an explicit task ID is provided to the student along with the state, or the student must infer the task itself. The existing multi-task PD methods are discussed in Section 3.1. Multi-task PD enables the student to perform well on all tasks while sharing common structure and separating task-specific properties. PD is particularly useful in avoiding negative transfer [40], whereas training a multi-task agent from scratch can negatively impact performance [61]. Intuitively, this robustness to negative transfer is because the student learns from pre-trained teachers that are already optimized for each task. By focusing on replicating the correct behaviors from these expert policies, the student avoids interference between tasks that might occur if trained jointly from the start. This allows the student to leverage shared knowledge without negatively impacting performance on the individual tasks.

3

Related Work

In this chapter, we review previous work relevant to our research. We begin by discussing multi-task PD methods. We then cover offline MTRL algorithms, which focus on training multi-task agents using pre-collected data without further interaction with the environment. Next, we explore data-sharing techniques, which aim to improve single-task policies by leveraging offline data across multiple tasks. Finally, we explain policy sharing, which also attempts to enhance single-task policies but through selective behavior sharing.

3.1. Multi-Task Policy Distillation

The paper that introduced PD [51] investigates the effectiveness of three distillation loss functions. The authors maintain Deep Q-Networks (DQNs) [44] for several Atari games. The goal is to transfer knowledge from all of these teacher DQNs to a single student DQN capable of performing all of the games. The first loss function is the Negative Log Likelihood (NLL) between the best actions of the teachers and the student. The second loss function is the Mean Squared Error (MSE) between the Q-values of the teachers and the student. The third loss is the Kullback-Leibler (KL) divergence on the softmax of the Q-values of the teachers and the student. Experiments point out that the KL-divergence is the most effective loss function in their particular setup. Furthermore, results indicate that in some situations the student can outperform its teachers as it can have a regularizing effect.

Actor-Mimic [48] is another distillation method operating in the discrete action space with DQNs. It showed similar results by transforming expert Deep Q-Networks (DQNs) into actor networks and transferring their knowledge to the student utilizing a cross-entropy loss. However, other than the original PD paper, the authors find that negative transfer occurs in some experiments.

Distral [61] is another successful method that introduces a form of multi-task exploration. Similar to PD and Actor-Mimic, it distills common behavior from the teachers to the student. However, it also uses the student to regularize the teachers, allowing them to explore the environment with newly learned behaviors from other tasks. It is important to note that, unlike other PD methods, Distral does not provide an explicit task ID to the student policy. The reason for this is to prevent the student from overfitting to its teachers. If the student overfits, it would simply replicate the teacher's actions, offering no exploration. Distral aims to enhance the performance of the teacher policies and deploy them separately after training. This is different than our objective to deploy a compact multi-task student. Nonetheless, we will explore ways of multi-task exploration similar to Distral.

KTM-DRL [70] is specifically focused on the continuous control setting. It uses TD3 agents and performs distillation using the MSE loss between the critics of the teachers and the student. Moreover, it introduces hierarchical experience replay, which picks transitions from each task's replay buffer to balance the skills learned across tasks. After distillation, the student is finetuned by interacting with all the tasks alternately, which is also a form of multi-task exploration. Results show that the student is capable of matching and sometimes slightly exceeding the performance of its teachers.

DROID [29] performs policy and reward distillation specifically applied to driving Mars rovers. It leverages demonstrations of expert human drivers and is capable of outperforming competing learning-from-demonstration methods. Nonetheless, it does not match the performance of the expert demonstrations.

However, in all of these studies expert teachers are assumed to be provided and utilized to collect data online. In this work, the aim is to investigate if the high performance of policy distillation translates to the offline MTRL setting, where we exclusively have access to previously collected offline data by an unknown behavior policy.

3.2. Offline Multi-Task Reinforcement Learning

In this research, we focus on offline MTRL. In this setting, the goal is to learn a single multi-task policy using only previously collected data from each of the tasks. No interaction is allowed with the environment before deployment.

Skills Regularized Task Decomposition (SRTD) [74] is a method proposed for this purpose. SRTD addresses the multi-tasking problem by decomposing tasks into subtasks, referred to as skills, which are short-term state-action sequences. It maps both tasks and skills into a shared latent space. This latent space is used to find the relationships between tasks, enabling the sharing of overlapping skills. Additionally, the offline data is augmented by sampling imaginary trajectories from the latent space. However, this approach heavily relies on the quality of the learned latent representations. If the latent space fails to accurately capture the commonalities and differences between tasks, SRTD may struggle to correctly identify shared skills, potentially leading to negative transfer.

Other methods, such as Pre-Training for Robots [36], MT-Opt [32], and Q-Transformer [12], focus on learning a multi-task policy by using large amounts of data. This data is generated from simulations or real-world demonstrations at a large scale. These approaches aim to improve policy learning with high-capacity models. In contrast, our work focuses on optimizing performance with fixed and relatively small datasets, without the possibility of gathering more data. Moreover, none of these techniques utilize policy distillation, which will be investigated in this work.

3.3. Data Sharing

Data sharing techniques focus on relabeling and sharing offline data across related tasks. Instead of training each task on its own dataset, all available data is used collectively to learn better policies for all tasks.

Conservative Data Sharing (CDS) [76] finds that naively sharing all data can improve learning but may degrade performance when it exacerbates the distribution shift between the behavior policy and the learned policy. To address this, CDS selectively shares data based on a conservative Q-value, balancing the quality of the data with the degree of distribution shift. Experiments show that CDS consistently performs better than without data sharing.

Uncertainty-based Data Sharing (UTDS) [4] argues that CDS is sensitive to a large distribution shift because the offline RL algorithm constrains the learned policy to stay close to the behavior policy. Instead, UTDS shares all data and applies pessimistic updates based on the uncertainty of state-action pairs, which mitigates the negative effects of suboptimal behavior policies. Results show that UTDS significantly outperforms CDS for random behavior policies and achieves comparable performance with more optimal behavior policies.

Unlabeled Data Sharing (CDS-zero) [75] addresses the scenario where the reward functions are unknown, whereas CDS and UTDS assume known reward functions for data relabeling. CDS-zero shows that simply assigning a reward of 0 to data from related tasks can still yield surprisingly good performance. However, this approach has limitations, especially when the shared data is highly relevant to the target task.

All of these methods utilize offline datasets from multiple related tasks to improve single-task policies. While this work is aimed at learning a multi-task policy, we explore how data sharing can be utilized as an extension to our approach in Section 6.4.2.

3.4. Selective Behavior Sharing

Q-Switch Mixture of Policies (QMP) [80] introduces selective behavior sharing across related tasks. Similar to the Distral method [61], it encourages multi-task exploration. However, while Distral achieves this by regularizing with a shared policy, QMP directly explores actions from the policies of other tasks. This approach is intended to avoid bias toward the average behavior of all tasks. Experiments show that QMP accelerates learning and improves task performance. Unlike our work, QMP does not perform policy distillation and is focused on learning single-task policies in the online setting. Nonetheless, we investigate whether the Q-switch mechanism can be adapted as an extension to our approach, as discussed in Section 6.4.3.

4

Approach

In this chapter, we describe two main online PD approaches from related work and explain how we extend them to operate in the offline setting. We also discuss the differences between the online and offline approaches. Following this, we introduce the experimental setup. Throughout the rest of the paper, we will explore different instances of our approach and specific adaptations, and we will conduct experiments within the same experimental setup.

4.1. Online Approaches

To motivate our choice of approach, we will discuss two main online approaches: Policy Distillation (PD) [51] and Distral [61].

The original online multi-task PD setup [51] is shown in Figure 4.1. Expert teachers are assumed to be provided and are deployed online to gather transitions. These transitions are stored in a separate replay memory for each task. Distillation begins by sampling transitions from the replay memories. The state and task ID (or game label) are fed into the randomly initialized student policy network. The student's output is then compared to the transition's target output using a distillation loss function. This loss is used to optimize the student network. As described in Chapter 3, this approach is capable of learning a student that can match or even exceed teacher performance.

A limitation of this setup is that there is no multi-task exploration. Knowledge from the teachers is distilled to the student, but the student never has the chance to explore the newly learned behaviors in the context of other tasks. Distral [61] introduces a form of multi-task exploration by regularizing the teachers with the student policy. This is displayed in Figure 4.2. It is important to note that Distral's

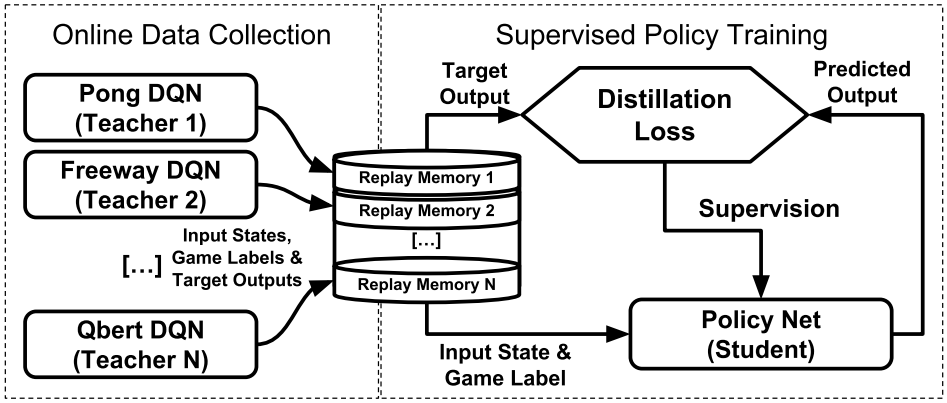


Figure 4.1: The original online multi-task PD setup [51]. The teachers are named after the Atari game they were trained on, but the approach can be applied to any domain consisting of related tasks.

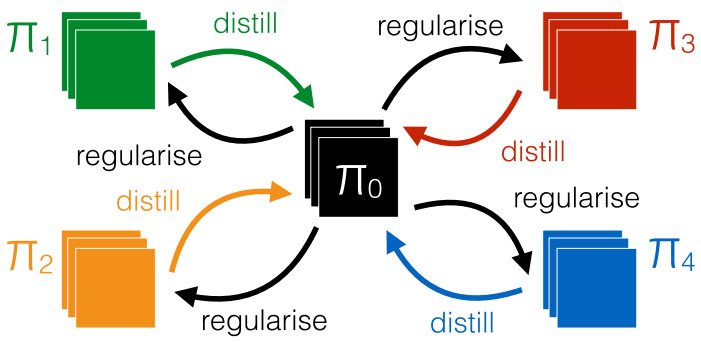


Figure 4.2: The Distral setup [61] distills knowledge from the teacher policies (colored) to the student policy (black), where the teachers are regularized by the student policy.

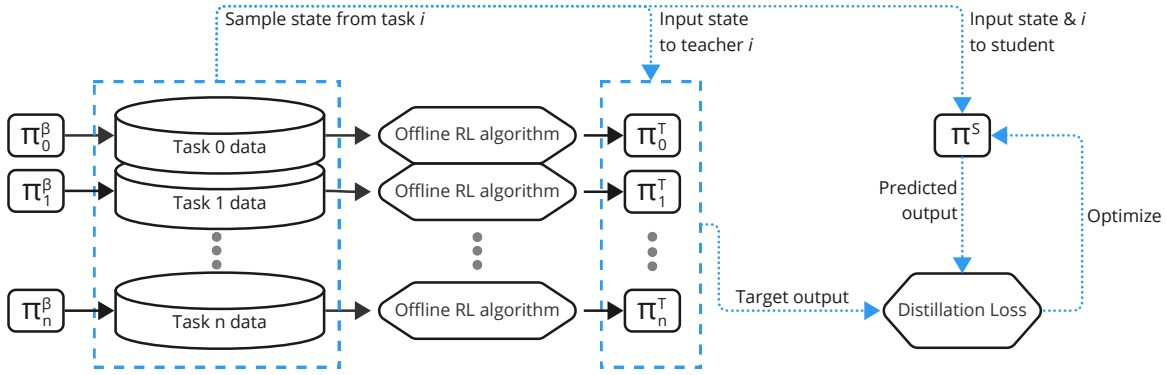


Figure 4.3: An overview of the offline PD approach, where the blue dashed lines represent the distillation process.

multi-task policy does not receive a task ID. This is because regularization would have negligible effect if the multi-task policy overfits to the task ID of its teachers. Instead, not providing a task ID encourages some randomization in the action selection from the different teachers, thereby allowing the student to explore behaviors from all teachers across tasks more effectively.

4.2. Offline Approach

Having explained the original PD (Figure 4.1) and Distral (Figure 4.2) approaches, we now introduce our offline approach. As mentioned earlier, the main difference between PD and Distral is the presence of multi-task exploration. However, in the offline setting, exploration is not possible, as there is no interaction with the environment before deployment. Instead, all transitions are collected beforehand by a behavior policy. This eliminates the benefit of using Distral's approach for our purposes, so we build upon the original PD approach.

We aim to extend the original PD approach in the simplest way possible. We will investigate its effectiveness and propose adaptations where necessary. Our approach consists of learning teacher policies for each task from offline data and then distilling these teachers into a single multi-task student policy. Figure 4.3 provides an overview of our offline approach. We will refer to this framework as MOP (Multi-task Offline Policy distillation).

The first step in our approach involves learning a teacher policy for each task. For every task, a behavior policy collects data from the environment, which is then used by an offline RL algorithm to learn the corresponding policy. The outcome of this step is a teacher policy for each task. Ideally, these teacher policies outperform their respective behavior policies, providing a strong foundation for the next phase.

The second step focuses on distilling the learned teacher policies into a single multi-task student policy.

Distillation begins by sampling states from the previously collected data of each task. These states are fed into both the corresponding teacher network and the randomly initialized multi-task student network, along with the task ID. The outputs of the teacher and student are compared using a distillation loss function. Potential loss functions will be explored in Section 5.1.3. This loss is then used to perform an optimization step on the student network. The objective is to minimize the loss, which occurs when the student accurately predicts the same outputs as the teachers for the same states and tasks.

4.3. Online and Offline Comparison

Our offline approach would be equivalent to the online case studied in previous work [51], if two conditions are met:

1. All behavior policies are of expert level.
2. The learned teacher policies are equivalent to their corresponding behavior policies.

These conditions ensure that the assumptions of online PD are satisfied. The first assumption is that expert teachers are provided. This holds, because the behavior policies are of expert level and the teacher policies match them. The second assumption is that the teachers are deployed in the environment to collect transitions. This also holds, since the behavior policies collect the data and the teacher policies are equivalent to the behavior policies, making it equivalent to the teacher policies collecting the data.

However, such an ideal scenario is unlikely in the offline setting. Behavior policies are often not of expert level and can even be random [17]. Additionally, offline RL algorithms are typically designed to improve upon the behavior policy [38], meaning the learned teacher policy likely differs from the original behavior policy.

The discrepancy between the teacher and behavior policies can be particularly problematic. The teacher policy is learned from the offline data, which was collected by the behavior policy. Since no new samples can be collected in the offline setting, the teacher policy is constrained to stay within the state distribution generated by the behavior policy. However, the teacher policy can still learn to behave differently within that state distribution. This would lead to a situation where the teacher's distribution of states (the states it would naturally visit) differs from the state distribution of the offline dataset.

This mismatch is known as a state distribution shift. Although the teacher policy is learning from the offline data, the distribution of states it would encounter during deployment may be different from the states present in the offline dataset. As a result, the student policy, which is trained to mimic the teacher using the offline data, may struggle to learn the correct behavior in situations that the teacher would naturally encounter. If those situations are underrepresented in the offline dataset, the student may not be able to generalize well to deployment. This can significantly degrade the performance of the student. This challenge will be further explored in Section 5.2.

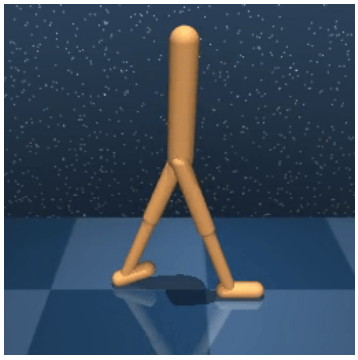
4.4. Experimental Setup

We will investigate the effectiveness of our approach under the same experimental setup throughout this work. In this section, the experimental setup is described by introducing the environment and explaining how the offline data was collected.

4.4.1. Environment

The environment used in the experiments is DeepMind Control Walker [59], as shown in Figure 4.4. It consists of a planar body with 7 segments and 6 joints. The state space is continuous and numerical, representing the orientations and velocities of the body parts. The action space is also continuous, representing the torque applied to the joints. The transition dynamics are governed by the MuJoCo physics simulator [62].

We have chosen this environment, because offline RL problems are more apparent in continuous state and action spaces. Continuous spaces are often more challenging than discrete ones since it is infeasible to visit every state in a continuous domain [17]. This forces an agent to generalize beyond the states and actions present in the offline dataset. This generalization introduces additional complexity in offline RL, because the agent must learn the correct behavior for unseen states based on previously collected



(a) Stand expert

(b) Walk expert

Figure 4.4: Examples of expert policies in the stand and walk task.

data with limited coverage. Without the ability to interact with the environment and to collect new samples, the agent relies on its ability to generalize, which can lead to performance degradation in underrepresented areas of the state space. This makes continuous environments particularly suitable for evaluating offline RL approaches.

On the other hand, in environments with discrete state and action spaces, offline RL problems are less challenging. When the offline dataset has low coverage for certain state-action pairs, the algorithm can simply ignore these parts of the state-action space or assign low value to unseen pairs. Discrete spaces are easier to tabulate, so even if the dataset is incomplete, the algorithm can easily avoid the missing areas. Furthermore, having discrete states allows for estimating state distributions, which can be used to reweigh samples and mitigate the effect of a state distribution shift. So offline RL challenges generally vanish in discrete environments.

4.4.2. Tasks

In our experiments, we use two tasks: standing and walking, each with its own reward function. The standing task rewards an upright torso and a minimum torso height, while the walking task includes an additional reward component for forward velocity. Expert policies for both tasks are shown in Figure 4.4. These policies are considered expert because they achieve high values. In both tasks, the torso remains upright and elevated from the ground. Additionally, the walking expert successfully moves forward.

The tasks share common structure, such as the position and orientation of the torso and interpretation of features from 'raw' states. Negative transfer could occur if the stationarity in the stand task is applied to the walk task. Similarly, forward bending of the torso for balance in the walk task could be incorrectly transferred to the stand task.

The common structure that a multi-task agent should share across tasks exists at multiple levels. The most straightforward common structure is at the trajectory level, meaning that parts of the state-action trajectories overlap across tasks. In those overlapping parts, the multi-task agent should take the same action for the same state, regardless of the task ID. In the standing and walking tasks, this corresponds to the initial part of the trajectory where the agent needs to stand up from the ground. Whether the task is to stand still or to walk, the agent must first stand up.

Additionally, common structure may exist at the computational level. At this level, the states and actions might differ entirely, but certain aspects of the policy computation—mapping a state to an action—can be shared across tasks. This can be thought of as underlying skills, rather than a one-to-one transfer of behaviors. In the standing and walking tasks, this would correspond to the skill of keeping the torso upright. While the states are different, as the leg segments have different orientations when standing or walking. The actions differ as well, since standing involves mostly stationary joints, while walking requires dynamic joint movements. However, keeping the torso balanced is a shared skill across both tasks, regardless of the task-specific requirements. We will further discuss and formalize common structure in Chapter 6.

We have selected only two tasks to simplify the multi-tasking setup. Previous work [4] extends this

approach with two additional tasks: running and flipping. Running is similar to walking but requires a higher forward velocity, while flipping involves rotating the torso vertically. For flipping, the common structure is less clearly defined since the torso does not need to remain upright, making it less suitable for multi-tasking.

4.4.3. Data Collection

As shown in Figure 4.3, our approach consists of behavior policies for each task that gather data from the environment. This offline data serves as the starting point for the offline RL algorithms to learn teacher policies and is also used to sample states during distillation. We will now define the behavior policies and explain how we employ them to collect data in our experimental setup.

The behavior policies are established by training TD3 [19] agents for each task. We utilize TD3 because it effectively learns expert policies for both tasks. The specific configurations of the TD3 agents are provided in Appendix A, following guidelines from prior work [5].

By using early stopping, we are able to establish different levels of behavior policies. This approach creates a realistic offline RL scenario, where the behavior policies can have different proficiencies. Medium-level behavior policies are obtained by early stopping at 1 million gradient steps, while expert-level policies are achieved by training until convergence at 2 million gradient steps. Initially, we aimed to include random-level behavior policies, which are typically used for broader exploration [17]. However, in our environment, a random policy fails to coordinate the body parts, resulting in no exploration at all. As a result, we excluded it from the experiments. The details of the policies are listed in Table 4.1.

Policy	Gradient Steps	Final Episode Reward
Stand (Medium)	$1 \cdot 10^6$	463
Stand (Expert)	$2 \cdot 10^6$	979
Walk (Medium)	$1 \cdot 10^6$	691
Walk (Expert)	$2 \cdot 10^6$	907

Table 4.1: Details of the medium and expert behavior policies for the stand and walk tasks. The final episode reward represents the reward obtained by the final policy averaged over 10 episodes.

Based on the medium and expert behavior policies, we generated three offline datasets consisting of 1 million transitions per task:

- **Expert**: rollouts of the expert policy
- **Medium**: rollouts of the medium policy
- **Medium-replay**: all 1 million transitions observed in the replay buffer during the training of TD3 up to the medium level

The expert and medium datasets are logically derived from the established behavior policies. Additionally, we decided to include a medium-replay dataset, which typically has broader coverage as it contains all exploratory transitions encountered during learning. On one hand, this wider variety in the data can make it easier for an offline algorithm to learn a policy, as the agent is exposed to more diverse states. On the other hand, it can exacerbate the state distribution shift, since the distribution of states in the dataset is likely much broader than the state distribution of the learned policy. We will discuss the consequences of this in Section 5.2.

5

Offline Policy Distillation

This chapter addresses the first research question: how does PD operate effectively in the offline MTRL setting? First, we will establish the standard setup by exploring different instances of the approach described in Section 4.2. Then, we will examine the impact of a state distribution shift, a major challenge in offline RL, on this standard setup.

5.1. Standard Setup

In Section 4.2, the general offline approach was discussed. Now, we will refine this by exploring specific instances of the method. The result will be a standardized setup that will be used throughout the rest of this work.

5.1.1. Learning Teacher Policies

The first step of the method involves learning teacher policies for each task from previously collected data using an offline RL algorithm. It is important to emphasize that the choice of algorithm is flexible, as long as it produces an effective teacher policy in the target environment. This means that if a more effective algorithm is introduced in the future, it can be easily integrated into our approach. Therefore, we will not conduct an in-depth analysis of algorithm selection; the goal is simply to find an algorithm capable of matching or surpassing the performance of the behavior policy.

For our experimental setup, which requires continuous control, we use Pessimistic Bootstrapping for Offline RL (PBRL) [5]. The specific configurations of the PBRL agents are provided in Appendix A, following guidelines from prior work [5].

For each offline dataset, we trained a new PBRL agent for 1 million gradient steps. The resulting policies closely match or slightly exceed the performance of their corresponding behavior policies. The results for the medium datasets are shown in Figure 5.1. The complete results can be found in Appendix C.

5.1.2. Justification For Policy Distillation

Before investigating how to perform the distillation most effectively, it is important to justify the need for it. This is best demonstrated through a naive multi-task implementation of PBRL, which is displayed in Figure 5.2. PBRL is an actor-critic method, where the actor represents the policy and the critic represents the Q-value function. Since the goal is to develop a multi-task policy, the actor is modified to receive a task ID as input and is used across all tasks. However, because the reward functions differ between tasks, a separate critic is maintained for each task. The training procedure alternates between sampling batches from the stand and walk datasets.

The results, shown in Figure 5.3, indicate that this naive multi-task implementation of PBRL yields low rewards and flat learning curves. The multi-task agent is identical to the single-task agents, except that the actor was adapted to handle multiple tasks. While there are many potential ways to modify this naive multi-task agent to improve performance, such changes would no longer reflect a naive

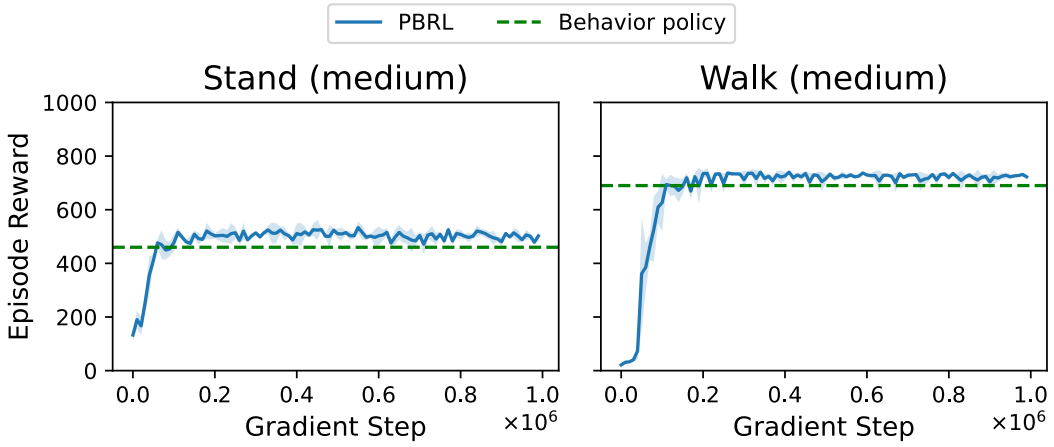
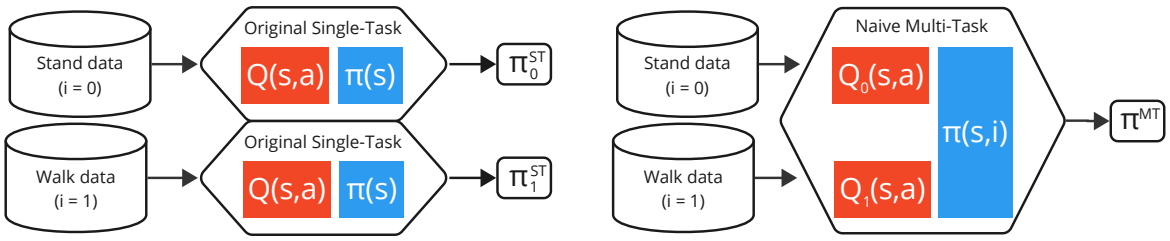


Figure 5.1: Results of PBRL [5] on the medium offline datasets, showing performance of 3 independently trained agents, each using a different random seed. The shaded area represents the 95% confidence interval.



(a) Original single-task

(b) Naive multi-task

Figure 5.2: A comparison of the original single-task PBRL agents and our naive implementation of a multi-task PBRL agent. The multi-task agent maintains separate critics for each task, while its actor is modified to handle both tasks.

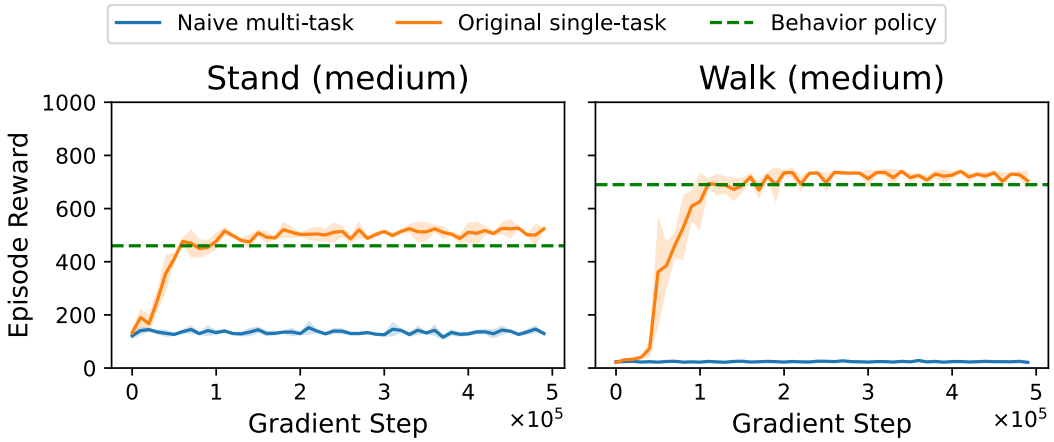


Figure 5.3: Results of the original single-task PBRL and our naive implementation of multi-task PBRL, each trained from scratch with 3 independent seeds on the medium offline datasets. Shaded areas indicate 95% confidence intervals.

implementation. We conclude that introducing a multi-task actor prevents the agent to effectively learn either task. This suggests that negative transfer is occurring, and a more sophisticated multi-tasking approach is required. In the following, we will investigate whether PD can address this issue.

5.1.3. Distillation Loss

The goal of PD is to enable the student network to replicate the outputs of the teacher networks for the same states and tasks. To achieve this, it is required to use a loss function that properly encourages this objective. We will consider three potential loss functions: Mean Squared Error (MSE) loss, Q-value loss, and Kullback-Leibler (KL) divergence.

Before discussing each of these loss functions, it is important to note that during optimization, all teacher agents are *frozen*. This means that while gradients can flow through them, their weights remain constant. This is because PD operates like supervised learning, where the input state serves as the 'features' and the teacher's output as the 'label'. If the teachers' weights were updated during training, the labels would change, affecting the ground truth. Freezing the teacher agents prevents this from happening.

Mean Squared Error Loss

The first approach minimizes the MSE between the actions generated by the teachers and the student. This method is straightforward and only requires access to the actor of each teacher. The corresponding loss function is:

$$\mathcal{L}_{MSE} := \frac{1}{m} \sum_{i=0}^{m-1} \mathop{\mathbb{E}}_{s \sim \mathcal{D}_i} \left[\left\| \pi_i^T(s) - \pi^S(s, i) \right\|_2^2 \right],$$

where m is the total number of tasks, i is the task ID, $\mathcal{D}_i$ is the offline dataset for task i, $\pi_i^T : \mathcal{S} \to \mathcal{A}$ is the actor network of the teacher for task i, and $\pi^S : \mathcal{S} \times i \to \mathcal{A}$ is the student's actor network.

Q-value Loss

While the MSE loss considers the optimal action for a given state, it ignores other actions that might be nearly as good. To address this, an alternative approach is to apply a Q-value loss, which aims to learn a multi-task actor that maximizes the teachers' critics. The loss function in this case is:

$$\mathcal{L}_Q := -\frac{1}{m} \sum_{i=0}^{m-1} \mathop{\mathbb{E}}_{s \sim \mathcal{D}_i} \left[Q_i^T(s, \pi^S(s, i)) \right],$$

where m is the total number of tasks, i is the task ID, $\mathcal{D}_i$ is the offline dataset for task i, $Q_i^T : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the critic network of the teacher for task i, and $\pi^S : \mathcal{S} \times i \to \mathcal{A}$ is the student's actor network.

Kullback-Leibler Divergence

However, the Q-value loss can be sensitive to differences in reward scaling across tasks. To mitigate this, and to consider more than just the single best action, we can use KL-divergence as the loss function. KL-divergence measures the difference between two probability distributions. To apply this, we modify the actors to be stochastic, returning the mean and covariance of a multivariate Gaussian distribution:

$$\pi_{Gauss,i}^T : \mathcal{S} \to \Delta(\mathcal{A}) := (\mu_i^T, \Sigma_i^T)$$
$$\pi_{Gauss}^S : \mathcal{S} \times i \to \Delta(\mathcal{A}) := (\mu^S, \Sigma^S),$$

where $\mathcal{A} \in \mathbb{R}^k$, $\mu \in \mathbb{R}^k$, and $\Sigma \in diag(\mathbb{R}^k)$.

Actions can then be sampled stochastically from the Gaussian distribution or deterministically by taking the mean:

$$a_{stochastic} \sim \mathcal{N}(\mu, \Sigma)$$

$$a_{deterministic} = \mu$$

The loss is computed on the sufficient statistics of the Gaussian distribution:

$$\mathcal{L}_{KL} := \frac{1}{m} \sum_{i=0}^{m-1} \mathop{\mathbb{E}}_{s \sim \mathcal{D}_i} \left[KL\Big(\mathcal{N}\left(\mu^S(s,i), \Sigma^S(s,i)\right) \| \mathcal{N}\left(\mu_i^T(s), \Sigma_i^T(s)\right) \Big) \right]$$

$$= \frac{1}{2m} \sum_{i=0}^{m-1} \mathop{\mathbb{E}}_{s \sim \mathcal{D}_i} \left[\operatorname{tr}\left(\left(\Sigma_i^T(s)\right)^{-1} \Sigma^S(s,i) \right) + (\mu_i^T(s) - \mu^S(s,i))^\top \left(\Sigma_i^T(s)\right)^{-1} (\mu_i^T(s) - \mu^S(s,i)) - k - \ln \frac{|\Sigma_i^T(s)|}{|\Sigma^S(s,i)|} \right] \text{[16]},$$

where m is the total number of tasks, i is the task ID, $\mathcal{D}_i$ is the offline dataset for task i, $\mu^S(s,i) := \left[\pi^S_{Gauss}(s,i)\right]_1$, $\Sigma^S(s,i) := \left[\pi^S_{Gauss}(s,i)\right]_2$, $\mu_i^T(s) := \left[\pi^S_{Gauss,i}(s)\right]_1$, $\Sigma_i^T(s) := \left[\pi^T_{Gauss,i}(s)\right]_2$, and k is the number of dimensions in $\mathcal{A}$.

The KL-divergence approach not only avoids issues with reward scaling but also accounts for the full distribution of possible actions rather than just the most likely one.

Results
Experiments in the Walker environment indicate that, somewhat surprisingly, the MSE loss performs best. The results for the medium datasets of the stand and walk tasks are displayed in Figure 5.4.

First, let us examine the teacher policies. The stochastic teachers perform worse than the deterministic ones. This is expected, as we adapted the PBRL algorithm to learn stochastic teachers, even though it was originally designed for learning deterministic policies. While we could replace PBRL with another offline RL algorithm to improve performance, this is not our primary objective. The choice of offline RL algorithm remains flexible, as our focus is on evaluating the effectiveness of the distillation process by comparing the performance of the multi-task student relative to its teachers.

Both the MSE and Q-value losses involve the deterministic teachers. MSE matches the teachers' performance, whereas the Q-value loss yields slightly lower values. Apparently, the more direct approach of the MSE loss makes optimization easier.

The KL-divergence involves the stochastic teachers. It fails to match teacher performance, particularly in the walking task. This approach is also less direct than MSE, as it minimizes distributions over actions rather than replicating the single best action. We suspect that the optimization is distracted by many irrelevant actions, leading to lower performance.

With the MSE loss successfully matching teacher performance, we have a solid baseline, which will be used for the remainder of this work.

5.2. State Distribution Shift

In the previous section, we established a standard setup for offline multi-task PD and demonstrated that it can successfully train a multi-task student to match the performance of its teachers on the medium offline datasets. However, the approach falls slightly short on the walking task with the medium-replay dataset (see Figure 5.5). As discussed in Section 4.4.3, the medium-replay dataset consists of all transitions that the TD3 agent encountered during training up to the medium level, including many exploratory samples that the medium policy is unlikely to encounter during deployment. In other words, the state distribution of this dataset is probably much wider than that of the medium dataset. This variety typically makes it easier for an offline RL algorithm to learn a good teacher policy. Nonetheless, the resulting teacher is trained to avoid low-reward transitions, which most likely causes its state distribution to be much narrower than the dataset it was trained on. Such a mismatch would result in a large state distribution shift between the teacher and the offline data.

As mentioned earlier in Section 4.3, a significant state distribution shift between the offline data and the learned teacher policy can negatively impact the distillation process. The offline dataset likely

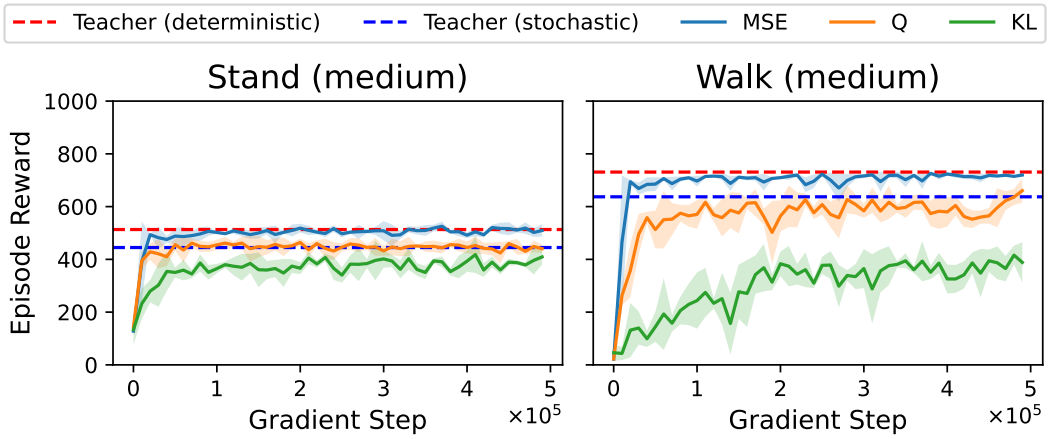


Figure 5.4: Results of the different distillation losses, each trained from scratch with 4 random seeds on the medium offline datasets. The shaded areas represent the 95% confidence intervals. MSE and Q are distilled from the deterministic teachers, whereas KL is distilled from the stochastic teachers.

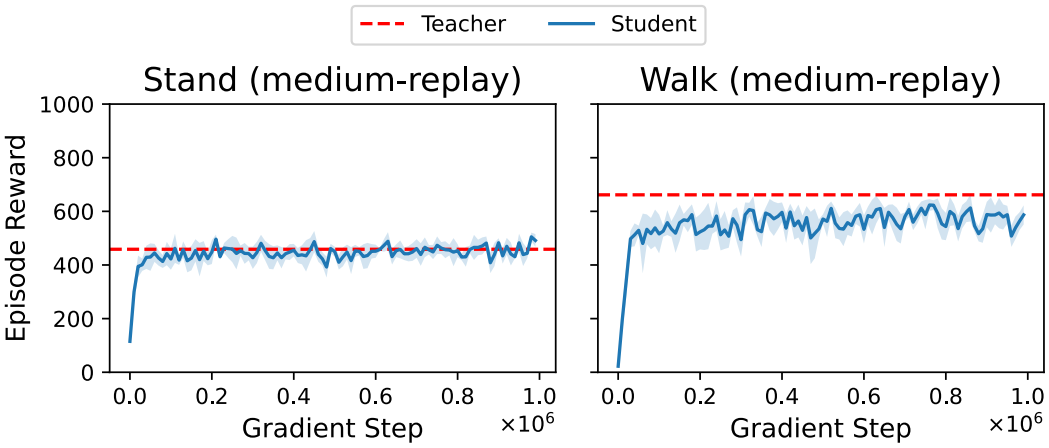


Figure 5.5: Results of MOP for the medium-replay datasets show that the student is unable to match teacher performance in the walking task. Note that the training time was doubled to 1 million gradient steps to verify whether the student would eventually catch up. The experiment was conducted over 4 random seeds and the shaded area represents the 95% confidence interval.

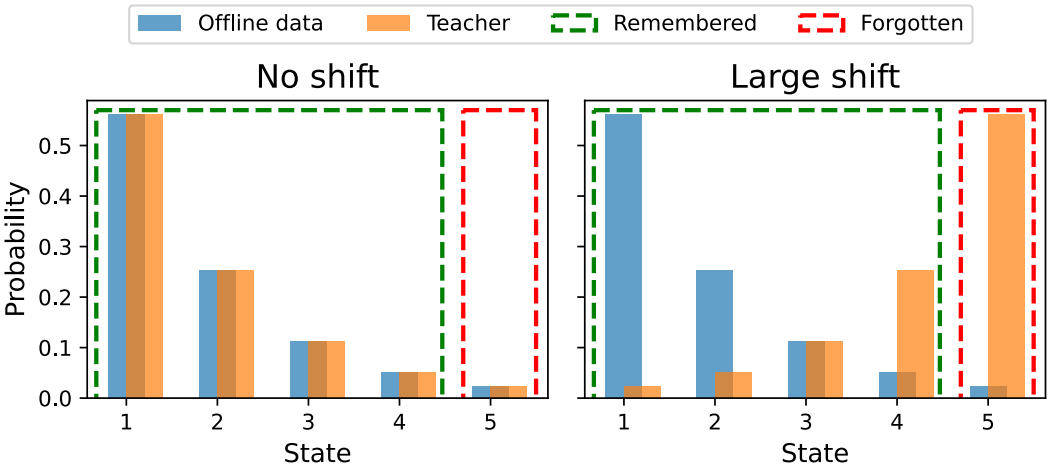


Figure 5.6: No distribution shift (left) and a large distribution shift (right) between the offline data and the teacher. Dashed lines indicate states remembered (green) or forgotten (red) by a student policy with a capacity to store 4 state-action pairs.

contains many states that the teacher would not encounter during deployment, making them irrelevant. Consequently, during PD, the student attempts to mimic the teacher's behavior in these irrelevant states, which may require more capacity in the student network. This observation leads to Hypothesis 1. In the remainder of this section, we will explore this hypothesis through a series of experiments to verify its validity.

Hypothesis 1 *A larger state distribution shift between the offline data and the learned teacher policy increases the capacity required in the student network.*

5.2.1. Hypothesis Clarification

To clarify the hypothesis, let us consider a simple example (displayed in Figure 5.6). In this example, there are 5 discrete, single-dimension states. The states in the offline data follow a negative Boltzmann distribution $p(s) = \frac{\exp(-(s-1))}{\Sigma_{x=1}^{5}\exp(-(x-1))}$. If the teacher's state distribution matches this, there is no distribution shift. However, if the teacher follows a positive Boltzmann distribution, $p(s) = \frac{\exp(s-1)}{\Sigma_{x=1}^{5}\exp(x-1)}$, a large distribution shift occurs.

The hypothesis suggests that a larger state distribution shift requires more network capacity. In other words, if the student network has limited capacity and there is a significant distribution shift, its performance will degrade. This occurs because the student's actions are learned based on states sampled from the offline data. With limited capacity, the student may forget what to do in states that have a low probability of being sampled.

In the simple example, suppose the network has the capacity to remember the correct action in at most 4 states. The fifth state will most likely be forgotten, because it has the lowest probability in the offline data distribution. If there is no distribution shift, this has minimal impact since the teacher only visits the fifth state 2% of the time. However, with the large distribution shift, performance degrades significantly because the student does not know what action to take in a state that the teacher visits more than 50% of the time. Increasing the network's capacity to handle all states would allow the student to remember the correct action for state 5, thereby improving performance. Thus, we hypothesize that a larger state distribution shift requires greater capacity in the student network.

Note that this example consists of a small, discrete space, making it feasible to tabulate and remember all states in practice. In contrast, our setup involves a large, continuous state space. It is impossible for all states to be represented in the offline data, let alone for the student to remember all of the states. Instead, the student must generalize over regions of the space, which increases the likelihood of 'forgetting' the correct actions in underrepresented regions.

5.2.2. Measure State Distribution Shift

The hypothesis arose from the assumption that the medium-replay dataset of the walking task has a large state distribution shift relative to the teacher policy learned from it. To support this assumption, we aimed to measure the KL-divergence between the state distributions of two datasets: the offline dataset collected by the behavior policy and the online dataset collected by the teacher policy. However, estimating densities in a continuous state space with 24 dimesions is challenging.

We implemented two methods: Kernel Density Estimation (KDE) [67] and Histogram Density Estimation (HDE) [18]. KDE with Gaussian kernels fails due to high dimensionality and insufficient samples (1 million per dataset). HDE, which divides the space into bins, is impractical as the number of bins grows exponentially with dimensionality; only three bins per dimension already requires over a Terabyte of memory. As a result, we were unable to produce reliable measurements and decided to omit them.

This limitation is a disadvantage of using such a complex environment. However, as previously discussed, if we were to use a simpler, toy environment with discrete states, many of the typical offline RL challenges would disappear. In such a case, the behavior policy could potentially visit all states, or the student network could memorize the best actions for every state in the offline data. To truly address the challenges of offline RL, we must evaluate the method in a complex environment where these issues are far more pronounced.

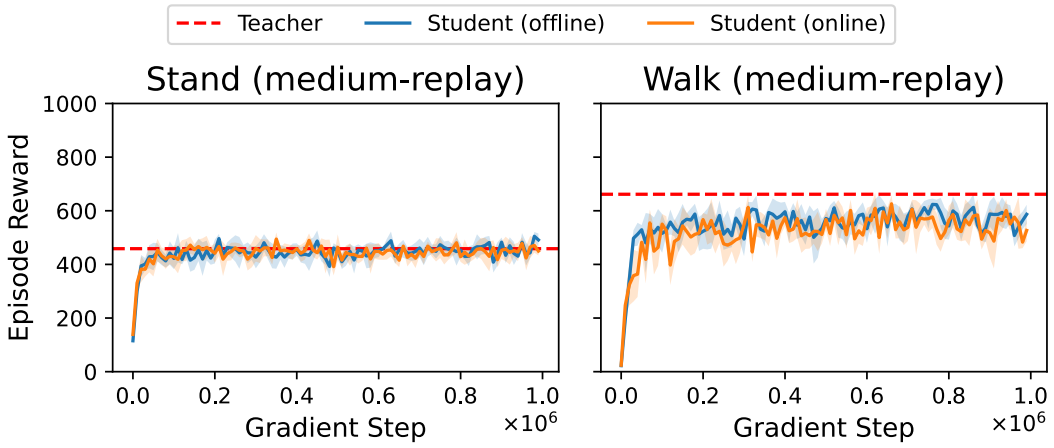


Figure 5.7: Results of MOP with online and offline state sampling on the medium-replay datasets, each trained from scratch with 4 random seeds. Shaded areas represent the 95% confidence intervals. Even with eliminated distribution shift (online), teacher performance is not matched in the walking task (right).

5.2.3. Eliminate State Distribution Shift

Another approach to verify the hypothesis is by comparing the results of offline and online PD. In this experiment, PD is run twice with identical settings, except for the dataset used for state sampling. In the offline setting, the offline datasets are used as before. In the online setting, an online dataset is created by deploying the teachers in the environment and collecting trajectories. The states in the online dataset correspond to those the teachers would naturally visit, thereby eliminating the distribution shift. If the hypothesis is correct, less network capacity should be required for online PD. In other words, it should yield a higher reward with the same capacity. Thus, it is expected that online PD will match teacher performance. Note that this would be similar to the other offline datasets, which are assumed to have smaller distribution shifts.

The results are shown in Figure 5.7. Contrary to expectations, online PD does not outperform offline PD in this setting. This suggests that the distribution shift is not the cause of the issue in the medium-replay walking dataset. Nevertheless, the hypothesis remains plausible, and further investigation is needed for verification.

5.2.4. Vary Network Size Under Artificial State Distribution Shift

We are not capable of measuring the distribution shift, as described in Section 5.2.2. However, we can create offline datasets with an artificial distribution shift to test the hypothesis. The goal is to verify whether a larger distribution shift leads to decreased robustness with smaller network sizes.

Composed Datasets

For this purpose, we compose four datasets for each task i:

- **Online** : 100% collected by π_i^T (the teacher policy for task i)
- **Noise**: 10% collected by π_i^T, and 90% collected by $\max(-1, \min(\pi_i^T + \mathcal{N}(0,1), 1))$
- **Random90**: 10% collected by π_i^T, and 90% collected by $\mathcal{U}(-1,1)$
- **Random99**: 1% collected by π_i^T, and 99% collected by $\mathcal{U}(-1,1)$

The Online dataset is collected by the teacher policy itself, so the state distributions are identical, resulting in no shift. Both the Noise and Random90 datasets contain 10% of trajectories collected by the teacher policy to ensure that the student has the opportunity to replicate the teacher's behavior. For Random99, this percentage is only 1%, which has the risk that the student ends up distilling on states irrelevant to the teacher, making it challenging to copy the correct behavior.

To introduce an artificial distribution shift, the Noise dataset includes 90% of trajectories collected by deploying the teacher policy while injecting Gaussian noise into the actions. The noisy actions are

clipped to ensure that they remain within the action bounds of -1 and 1. We expect this to cause a slight distribution shift, although we cannot measure it directly. Upon reviewing video recordings of the noisy policy being deployed in the environment, it becomes clear that the policy is still capable of performing the task, although highly disturbed by the noise. This suggests that many states will be encountered which the regular teacher policy would not, for example when the noise causes the body to fall while walking. This induces at least a slight shift between the state distributions.

The Random90 and Random99 datasets contain trajectories where the actions are sampled from a Uniform distribution within the action range of -1 and 1. These random actions fail to coordinate the joints, resulting in no effective exploration. Instead, the body remains on the ground, randomly moving its joints. These states are absent or highly underrepresented in the teacher policy's state distribution. We expect this to induce a significant state distribution shift. The shift for Random99 should be even larger than that of Random90, as the percentage of teacher trajectories is even lower.

Varying Network Size Experiment

We test MOP's performance on each of these datasets with varying network sizes. The original architecture of the student network consists of an input layer matching the state dimensions and an additional dimension for the task ID, two hidden layers with 256 neurons each, and an output layer matching the action dimensions. In this experiment, the number of neurons in the hidden layers is varied between 16 and 512. Following Hypothesis 1, we expect that the performance of the student trained on datasets with a larger distribution shift relative to the teacher policy will degrade more as the network size decreases.

Results

The results are shown in Figure 5.8. The Random90 and Random99 datasets are the least robust to smaller network sizes, which aligns with our initial expectations. However, contrary to expectations, the Noise dataset outperforms the Online dataset for the smallest network sizes. It is important to be cautious in making hard claims from these results, because the 95% confidence intervals highly overlap suggesting that the differences may not be statistically significant. Nevertheless, the results indicate that a larger state distribution shift does not impact performance as strongly as initially expected.

Remarkably, the Noise dataset appears to be the most robust to smaller networks. This suggests that increased variety in the training data can actually improve performance, even if it introduces a slight state distribution shift. We hypothesize that this occurs because smaller networks tend to take less optimal actions, lacking the capacity to remember all teacher behaviors. As a result, the agent is more likely to end up outside the teacher's state distribution. With more diverse training data, the agent is better equipped to handle these out-of-distribution states. On the other hand, when trained only on Online data, it may have no clear guidance for states slightly outside the teacher's distribution.

Furthermore, Random90 and Random99 achieve acceptable performance despite introducing substantial distribution shifts. Random90 nearly matches teacher performance for the largest network, which has 512 neurons per hidden layer. However, Random99 does not reach this level of performance. We suspect this may be due to either overfitting or the need for an even larger network.

Overfitting is a common issue in Supervised Learning (SL) [73]. It occurs when a model fits the training data too closely, but fails to generalize well to unseen test data. Overfitting can also happen in RL, even though training and testing occur within the same environment [79]. As discussed in Section 2.3.3, PD is a combination of RL and SL. In PD, the states sampled from the offline data can be considered as the training data, while the states encountered upon deployment serve as the test data. It is possible that the student network accurately mimics the teacher's actions in states used during distillation but struggles to generalize to unseen states during deployment.

The likelihood of overfitting increases when the training data is limited, and the network has many parameters [78]. For example, the Random99 dataset contains only a small subset of states (1%) that contribute useful behavior for distillation. Simply increasing the network size in such cases might exacerbate overfitting. This highlights an important difference between tabular policies and policy networks. In tabular policies, actions for each state are stored explicitly, which is feasible only in discrete state spaces. Expanding the table directly increases its capacity to remember more state-action pairs. However, increasing the size of a neural network enhances its ability to represent more complex

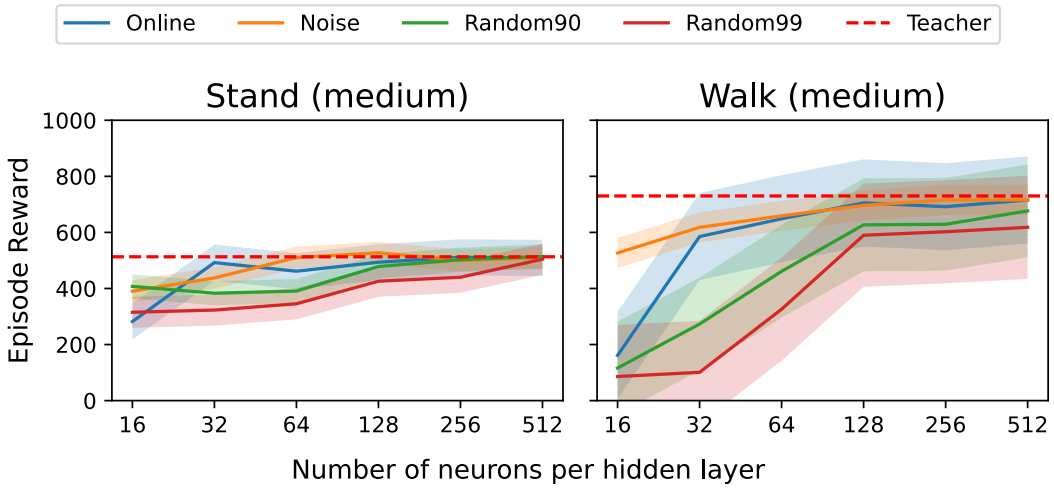


Figure 5.8: Results of MOP across varying network sizes and dataset types, each trained from scratch with 3 random seeds. Shaded areas represent the 95% confidence intervals.

functions. This does not necessarily imply that it can remember the best actions over a larger part of the state space. Instead, a larger network might overfit to the offline data, particularly when only a small amount of data is available.

However, we cannot completely exclude the possibility that increasing the network size even further might allow Random90's performance to eventually catch up with the teacher. Due to computational constraints, we did not test wider or deeper networks. Prior research [25] has shown that larger networks do not consistently lead to better performance in RL. In fact, deep RL agents can encounter instability during training with larger networks, which could be due to a rougher loss surface [54]. As a result, effectively utilizing larger networks in RL requires careful consideration of architecture and training methods [46]. However, these issues primarily arise when training RL agents from scratch, whereas we are distilling pre-trained teacher policies through SL. It would therefore be valuable for future research to investigate whether enlarging the network even further enables the student to match the teachers' performance under a large distribution shift.

In conclusion, the state distribution shift did not impact MOP's performance as much as initially expected. A slight shift can even be beneficial as it increases the variety in the data, which may improve out-of-distribution robustness. However, a very large shift could still degrade performance, even for larger networks, due to potential overfitting to the offline data. We recommend that future work explores the effects of further increasing the network size under significant distribution shifts. This should reveal whether overfitting occurs or if the student can eventually match the teachers' performance. Additionally, more sophisticated methods than simply increasing the network size should be explored to eliminate the risk of overfitting.

5.3. Discussion

In this chapter, we explored how to apply PD in the offline setting. First, we established a standard setup by examining specific instances of the approach described in Section 4.2. We demonstrated that the PBRL offline algorithm [5] is capable of learning single-task teacher policies that closely match or slightly exceed the performance of the behavior policies. Next, we found that directly multi-tasking through a naive multi-task implementation of PBRL suffers from negative transfer, failing to perform effectively on either task. PD, however, successfully mitigated this issue. We experimented with three different distillation loss functions: MSE loss, Q-value loss, and KL-divergence. The results showed that directly optimizing MSE loss yielded the highest performance, making it the standard distillation loss for the remainder of this work.

We then investigated the impact of a state distribution shift, a major challenge in offline RL, on our approach. Initially, we hypothesized that a larger state distribution shift between the offline data and

the teacher policy would require increased network capacity in the student network (Hypothesis 1). To test this hypothesis, we composed datasets with varying degrees of artificial distribution shifts, ranging from no shift (100% data collected by the teacher policies) to a large shift (1% data collected by the teacher policies). The results indicated that the hypothesis was partly correct and partly incorrect.

The hypothesis holds in that student performance generally declines when the network size is small and the distribution shift is large. However, a slight distribution shift can actually improve performance by increasing the variety in the offline data, which may help to enhance out-of-distribution robustness. Furthermore, simply increasing the network size does not necessarily resolve performance loss caused by a large distribution shift. This is because the combination of limited useful data and a network with many parameters may lead to overfitting [78].

Thus, we conclude that a state distribution shift does not significantly affect MOP's performance unless the shift is very large. Unfortunately, we cannot quantify what 'very large' means, as we are unable to measure the shift in our environment due to the large continuous state and action spaces. Increasing the size of the student network can help mitigate performance loss from a distribution shift, but this is not guaranteed to resolve the problem if the shift is too large. Exploring more sophisticated approaches to handle large state distribution shifts is an interesting direction for future research.

We leave the development of such an approach for future work, but would like to propose a promising idea. Typically, distribution shifts can be addressed by reweighting samples in the dataset. Importance Sampling (IS) is such a reweighting technique [58]. IS reweights samples by calculating the probability density ratio between the two distributions of interest, which requires the distributions to be known. This is not the case in our setting. However, we can use the ratio of Q-values to reweight samples in a similar fashion. A policy's Q-value reflects how it evaluates a state-action pair, indicating whether the policy is likely to visit (high value) or avoid (low value) that pair. Thus, the ratio of the Q-values of the behavior policy and the teacher policy could be used as an alternative to the probability density ratio. However, this assumes that the Q-value function of the behavior policy is known. While this is typically available in offline RL, it could be estimated through Behavioral Cloning (BC) [35] for example.

6

Capturing Common Structure Through Policy Distillation

In this chapter, we address the second research question: how can common structure be better captured through PD? As outlined in the introduction, our goal is to exploit common structure with two main objectives: improving performance and reducing required capacity. Ideally, common structure is highly exploited, allowing the size of the student network to be reduced while maintaining or even improving the performance.

We begin by formally defining common structure, which is essential for understanding the specific problem we aim to address. We define common structure at two distinct levels: the trajectory level and the computational level. Next, we introduce a measure to quantify the amount of shared computation, which serves as an indicator of how much common structure is captured.

We then explore methods to improve PD's ability to capture common structure. Initially, we focus on the computational level. Specifically, we investigate whether introducing an information bottleneck by constraining capacity can lead to increased sharing of common structure across tasks. Following this, we add a regularization term to the distillation loss to encourage shared computation.

Next, we shift our focus to the trajectory level. To address common structure at this level, we discuss the limitations of not having multi-task exploration. We argue that the reward functions must be known to enable multi-task exploration. Under this assumption, we propose two methods designed to share more common structure across tasks through multi-task exploration: Data Sharing (DS) and Offline Q-Switch (OQS).

The chapter concludes with a discussion of our findings and an answer to the second research question.

6.1. Common Structure Definition

Before attempting to better capture common structure, it is crucial to define what common structure actually is. In the literature, there is no single definition of common structure. This is because common structure is not a property of the environment, but rather a property of the behaviors represented in the policies. Common structure can be understood as behaviors that are shared across tasks. Essentially, when defining common structure, we are defining a solution to the multi-tasking problem. Since there are various ways to represent such solutions depending on the multi-tasking method, there are also multiple ways to define common structure.

We distinguish two types of common structure based on the level at which they occur: the trajectory level and the computational level. In this section, we provide an intuitive explanation of these types of common structure, attempt to formally define them, and link them to similar concepts found in the literature. Afterwards, we describe the expected challenges that we might encounter attempting to capture common structure.

Trajectory Level

Common structure at the trajectory level means that parts of the trajectories across tasks overlap. Within these overlapping parts, the agent should select the same action for the same state, regardless of which task it is trying to solve. To build intuition about this, let us consider a simple example in a grid world environment.

In this example, the environment is a 4×4 grid. Each cell in the grid represents a possible state, and the possible actions are *up*, *down*, *left*, or *right*. The transition dynamics are deterministic, meaning the agent moves in the direction specified by the action unless blocked by the boundary of the grid. For instance, if the agent is at the left boundary and attempts to move left, it will remain in the same cell. The agent always starts at the upper-left corner of the grid. The reward is 0 for every transition, except when the agent reaches the goal state, where the reward is 1. Each task is defined by a different position of the goal state, resulting in different reward functions. The objective for the agent is to reach the goal state as fast as possible, yielding the maximum discounted return.

An instance of such a grid world is shown in Figure 6.1. To reach the goal states from the top left, the first three state-action pairs are identical across tasks. These are marked in green, and we define them as part of the common structure. In the fourth state, a different action must be taken depending on which task the agent is solving. Therefore, the final parts of the trajectories are task-specific.

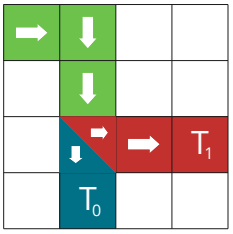


Figure 6.1: An example grid world with two targets.

We can formally define common structure at the trajectory level as a set of states in which the action chosen by the policy is identical across tasks. For two tasks this is defined by:

$$\pi(s,i) = \begin{cases} \pi(s,0) = \pi(s,1) & \text{if } s \in S_{\text{common}} \\ \pi(s,i) & \text{otherwise} \end{cases}$$

where s is the input state, i is the task ID, and S_{common} is the set of states where the policy actions are shared across tasks.

Several papers address multi-tasking from this perspective, even though they do not explicitly define common structure. For instance, Q-switch Mixture of Policies (QMP) [80] dynamically determines which actions to share across tasks in a state-dependent manner. The method ranks and selects the best action proposed by other tasks' policies for the same state. Over time, the agent learns which actions are beneficial to share across tasks and which should remain task-specific.

Furthermore, hierarchical approaches attempt to reuse parts of policies across tasks by defining lower- and higher-level policies [50]. Sharing specific lower-level policies across tasks is essentially equivalent to selecting the same action across common structure states irrespective of the task ID.

Lastly, data-sharing approaches [76, 4] focus on relabeling and sharing offline data across tasks to improve single-task policy performance. While these methods do not explicitly incentivize capturing common structure, they aim to optimize parts of trajectories by exploring state-action pairs across different tasks.

In our experimental setup, as described in Chapter 4.4, the common structure at the trajectory level corresponds to the agent standing up from the starting state. Regardless of whether the task is to stand or to walk, the agent must first get up from the ground. After this initial phase, the trajectories become task-specific: for the standing task, the agent must remain stationary, while for the walking task, it must start moving.

Computational Level

Common structure at the computational level can only occur in deep RL, where policies are represented by neural networks. This is because we assume that parts of the computation—mapping a state to an action—can be shared across tasks, even when the input state and output action are different. This is not possible if the policy is a table in which an action is stored for each state, because this does not entail any computation.

The idea behind common structure at the computational level is that there are underlying skills that can be reused across tasks. An intuitive example is the notion of 'ball feeling'. If a person is good at football, they can predict the movement of the ball and act upon it. When the same person plays tennis, many things are different: the size of the ball, the way the ball is played, the playing field, etc. However, this person is still capable of using the skill of predicting where the ball will go and how to hit the ball correctly. So the states and chosen actions can be completely different, but there is an underlying skill that is reused across tasks.

A possible way to formalize this is by using function compositions. The policy function can be composed of a set of functions, some of which can be shared across tasks. For two tasks, this is could be defined as:

$$\pi(s, i) = \begin{cases} h(f(s)) & \text{if } i = 0 \\ h(g(s)) & \text{if } i = 1 \end{cases}$$

where s is the input state, i is the task ID, h is a common function across tasks, and f and g are task-specific functions. Note that this is an example, many more functions can be involved and the order of how the functions are applied can also take many different forms.

Representation-based approaches consider common structure from this perspective. These methods aim to learn an abstraction of common structure in a representation and share it between tasks [8]. This shared representation enables policies to generalize more effectively across tasks.

As already mentioned in Chapter 4.4, common structure at the computational level can be thought of as the skill to keep an upright torso in our experimental setup. The encountered states and performed actions are different across tasks, since while standing the legs are stationary whereas while walking the legs are continuously moving. However, in both tasks the agent must keep an upright torso, which could possibly be enforced by shared computation across tasks in the multi-task student network.

Challenges
Having defined common structure from two perspectives, we can already foresee some challenges in capturing it through PD. At the trajectory level, it is possible that the teacher behaviors are unaligned, meaning their trajectories may never or only slightly overlap. During distillation, the student network copies the behaviors of the teachers, but the student cannot explore new behaviors across tasks because of the offline RL setting. If the behaviors are unaligned, this make it difficult for PD to capture any common structure at all. This will be addressed in Chapter 6.4.1.

At the computational level, enforcing partially shared computation across tasks can be challenging. The computation is not directly copied from the teachers, as the student is randomly initialized and only receives input states and target actions to minimize the loss. The student network independently learns how to compute actions from states. Therefore, an additional incentive should be introduced to encourage shared computation across tasks. We will experiment with this in Chapter 6.3.2.

6.2. Measuring Shared Computation

In Chapter 5, we demonstrated that MOP can learn student policies that match the performance of their teachers. Now, we aim to evaluate how much common structure these students capture. To achieve this, we design a technique to measure shared computation across tasks, which serves as an indicator of how much common structure is being captured. To the best of our knowledge, this is the first attempt to quantify the sharing of common structure across tasks.

We begin by discussing how the network's activation values can be leveraged for this purpose. Next, we explore measures to assess the similarity between these activation values across tasks. Lastly, we address the limitations of our measurement technique.

6.2.1. Activation Values

The student network used in our experiments consists of an input layer with 25 neurons (1 for the task ID and 24 for each state dimension), two hidden layers of 256 neurons each, and an output layer with 6 neurons corresponding to the action dimensions. The hidden layers use the ReLU (Rectified

(a) Separate task-specific subnetworks

(b) Highly overlapping task-specific subnetworks

Figure 6.2: An intuitive illustration of when there is no shared computation or high shared computation across tasks. The rectangles represent layers in the neural network with the number of neurons denoted inside. The blue and red colors represent two different tasks, where there is no overlap between activated neurons in (a) and high overlap in (b).

Linear Unit: $f(x) = max(0, x)$) activation function [23], which maps negative values to 0 and applies an identity function otherwise. The output layer leverages the hyperbolic ($f(x) = tanh(x)$) tangent activation function [33] to map the result between -1 and 1, which corresponds to the range of the action space.

Every time the network maps a state and task ID to an action, a neuron in a hidden layer is considered activated when the outcome of the ReLU is positive, and not activated when the outcome is 0. We are interested in understanding how these intermediate activation values correlate with the different tasks. Before diving into the exact measurement, let us first build some intuition about what would hypothetically occur within the student network when no common structure is captured compared to when a significant amount is shared.

Two scenarios are illustrated in Figure 6.2. If no common structure is shared, the student network might consist of two disjoint subnetworks, each dedicated to one task, with the task ID determining which subnetwork to activate (Figure 6.2a). In this scenario, there would be no shared computation across tasks, and there would be no advantage to maintaining a multi-task network. In fact, it would be equivalent to using two separate single-task networks and selecting which to use based on the task ID. Note that this is a hypothetical situation, and there is no direct indication that the network would develop two completely non-overlapping subnetworks. However, if such a case were to occur, it would imply that using a multi-task network offers no additional benefits compared to having separate single-task networks.

On the other hand, if a significant amount of common structure is exploited, the subnetworks would overlap considerably (Figure 6.2b). Many neurons would activate regardless of the task ID to process common structure. However, some neurons would still activate only for specific task IDs, reflecting task-specific properties.

While Figure 6.2 is a hypothetical illustration meant to build intuition, our goal is to quantify the correlation between neuron activation values and the task ID of the actual student networks. This comparison does not only consider whether a neuron is activated or not, but also takes into account the magnitude of its activation. We will later discuss how to numerically compare these activation values using a similarity measure. However, before doing so, it is essential to determine which activation values should be compared. There are two primary options for this: comparing per-state activation values or comparing activation values averaged over a task's own state distribution.

For both options, we deploy the student agent online to perform both tasks and let it collect transitions. We let it perform 1000 episodes per task. An episode consists of 1000 transitions, so it collects 1 million states per task. $\mathcal{D}_i^S$ denotes the dataset of transitions collected by the student in task i.

Per-State Activation Values

To compare the per-state activation values of the agent across tasks, we let the student infer an action for each state in the datasets and both task IDs: $\pi^S(s, 0)$ and $\pi^S(s, 1)$, where $s \sim \mathcal{D}_0^S \cup \mathcal{D}_1^S$. This allows us to

compare the activation values for the same state, but with different task IDs.

This approach reveals the influence of the task ID on the activation values of individual input states. A key limitation of comparing per-state activation values is that a state can implicitly indicate the task, even without the explicit task ID. This occurs because certain states might only be relevant to a specific task. In such cases, the student could output the same action for both tasks without a performance loss, since it would not encounter that state in the offline data or the environment for the other task. For these states, which are only relevant to one of the tasks, there could be high correlation across tasks. However, this correlation does not truly reflect the common structure shared between tasks. To capture the common structure more accurately, it is necessary to incorporate the state distributions of the student across both tasks.

Average Activation Values

To compare the activation values while accounting for the state distribution of each task, we can use the average activation values per task. As before, we utilize the datasets collected by the student in the environment. However, we only pass the states through the student network with the task ID corresponding to the task in which the state was encountered: $\pi^S(s, i)$, where $s \sim \mathcal{D}_i^S$. We then average the activation values per task and neuron.

This method accounts for the state distributions of each task, as it only considers the states the student would naturally encounter when performing that task. This is crucial to accurately compare the activation of neurons across tasks. However, a limitation is that averaging could disregard important details, as the activation values may fade out when averaged across all states. Nonetheless, this technique can still provide useful insights into which neurons are consistently active for each task. A high degree of overlap in active neurons across tasks may suggest that common structure is being effectively captured.

We can then plot these average activation values and compare them between the standing and walking tasks. Figure 6.3a shows this comparison for the student trained on the Medium-replay offline datasets. However, this visualization is difficult to interpret, as the network does not enforce any positional constraints on neurons with respect to the task ID. In other words, the network does not activate all the left-side neurons for the standing task and the right-side neurons for the walking task; instead, the activations are completely intertwined.

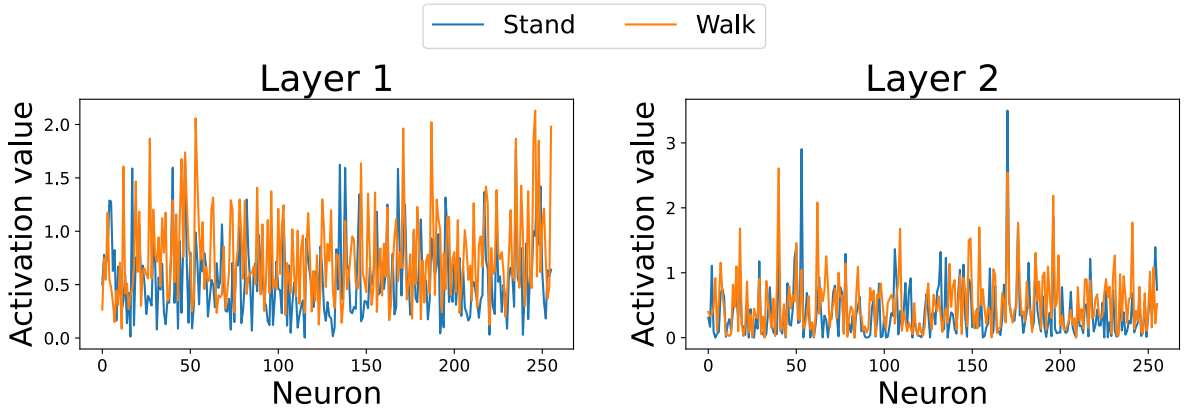
To make this visualization more interpretable, we sort the neurons based on their average activation values for the standing task. This is shown in Figure 6.3b. Since the neurons are sorted by the activation values of the standing task in ascending order, a positive correlation is indicated if the plot of the walking task shows an increasing trend. A negative correlation is indicated if the walking plot shows a decreasing trend. No correlation is indicated if there is no trend. The results show a slight correlation in the first layer and a strong correlation in the second layer. This suggests that the student is sharing computation across tasks and could mean that at least some common structure is captured.

6.2.2. Similarity Measure

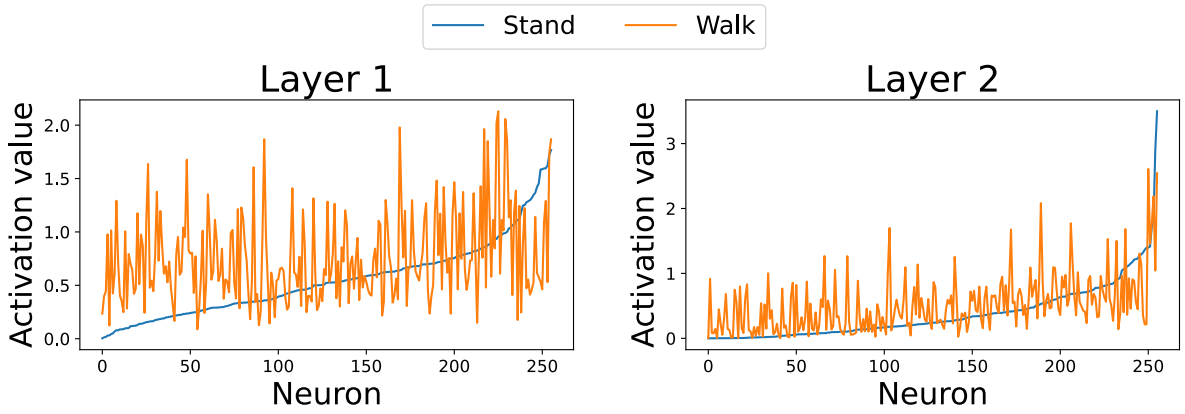
In the previous section, we explained how activation values can be compared across tasks, serving as an indicator of the common structure captured by the multi-task student network. Our goal is to quantify the correlation between activation values and task ID using a numerical metric rather than a plot. Therefore, we will use Kendall's τ as a similarity measure.

In the previous section, we plotted the average activation values across tasks (Figure 6.3). To make it visually interpretable and enable trend comparison between tasks, we sorted the neurons based on their activation values in the stand task. We believe that this ordering provides an intuitive way to compare neurons, as it reflects their relative activation rather than the absolute value. A commonly used measure for comparing such orderings is Kendall's τ.

Kendall's τ is widely used in information retrieval to measure the correlation between two ranked variables by comparing the number of concordant and discordant pairs [53]. An example of ranked variables is the output of search engines, which produce ranked lists of documents [72]. To assess the effectiveness of search engines, it is helpful to compare such ranked lists. In our case, Kendall's τ can be interpreted as ranking the neurons in each layer based on their activation values and then comparing the concordant and discordant pairs between tasks. Kendall's τ ranges from -1 (indicating negative



(a) Unsorted.



(b) Sorted by the average activation values of the standing task in ascending order.

Figure 6.3: The average activation values per neuron of the student network trained on the Medium-replay datasets after being deployed online to perform the standing and walking tasks.

correlation) to 1 (indicating positive correlation), with 0 representing no correlation. A high positive value would indicate that neurons in both tasks are similarly ranked, suggesting significant shared computation.

Formally, Kendall's τ is defined as:

$$\tau = \frac{2(C - D)}{N(N - 1)} \text{ [72]},$$

where C is the number of concordant pairs, D is the number of discordant pairs, and N is the number of objects in each ranked variable.

6.2.3. Results

The results of measuring shared computation are shown in Table 6.1. The similarity for per-state activations is significantly higher than for average activations. This is expected, as there may be high similarity for states encountered in only one of the tasks, as explained in Section 6.2.1. Furthermore, the per-state activations indicate more sharing in the first layer than in the second layer, whereas for the average activations, this is reversed. This observation makes sense because per-state activations are evaluated on the exact same input, leading to a correlation of 1 at the input level, which gradually diverges deeper in the network due to sequential transformations. In contrast, the average activations consider a distribution of input states, providing a measure of similarity that is less dependent on specific inputs.

We believe that the average comparison best reflects the purpose of our measurement, as it incorporates the state distributions of the tasks. Furthermore, the results indicate that activation values do not fade out against each other in our setting, which can be a risk when taking the mean. Therefore, we will use the average activation comparison as the primary technique to measure shared computation throughout this work. Only in cases where it is impractical to compute the mean of activation values, we will use the per-state comparison. When using the per-state comparison, this will be explicitly noted.

The results show that Expert X Expert shares the most. We suspect this sharing occurs because both teachers demonstrate correct behavior in states with common structures. Correct behavior is more likely to be aligned, making it easier for PD to merge. Conversely, Medium X Medium shares the least, which can be explained by the fact that suboptimal behaviors are less likely to be aligned. This observation will be discussed further in Chapter 6.4.1.

Furthermore, Medium-replay X Medium-replay shares a significant amount. This is expected, as the offline data includes all trajectories of TD3 trained up to the medium level. This implies that many similar states are present in the offline data for both tasks, leading to more similar state distributions. Consequently, this increases the likelihood of shared computation in these states.

Student Network (Walk X Stand)	**Per-state**		**Average**	
	τ **layer 1**	τ **layer 2**	τ **layer 1**	τ **layer 2**
Expert X Expert	0.57 ± 0.05	0.45 ± 0.06	0.16	0.36
Medium X Medium	0.53 ± 0.06	0.37 ± 0.08	0.067	0.15
Medium-replay X Medium-replay	0.54 ± 0.06	0.36 ± 0.07	0.14	0.36

Table 6.1: Results of per-state and average activation values comparison with Kendall's τ. The per-state values represent the mean and standard deviation over all state comparisons, while the average values are single-point comparisons of mean activation values.

6.2.4. Limitations

As mentioned earlier, this is the first attempt, to the best of our knowledge, to quantify the sharing of common structure across tasks. We recognize that our measurement technique may not fully capture the extent of shared structure in all cases. Therefore, we discuss its limitations and aspects that should be addressed in future work.

Comparing Different State Distributions

First of all, we compared activation values per-state and on average. The per-state comparison has a limitation: activation values are compared between tasks, but some states may be encountered in only one of the tasks. When inferring an action for these states with a switched task ID, it is possible that the computation remains highly similar. However, this similarity does not necessarily indicate shared behavior across tasks. Instead, it may occur because the state-action pair is irrelevant for one of the tasks, as it is never encountered. The network could produce any action in these states without degrading performance. By averaging across all states in a task, only the relevant states are considered for each task. This makes it a more appropriate method for accounting for task-specific state distributions. Nonetheless, this approach also has a limitation: averaging can cause activation values to fade out against each other. It would be interesting for future work to investigate other ways of comparing activation values, while accounting for different state distributions across tasks.

The concept of comparing activation values relates to prior work that examines the similarity of representations learned by different neural networks. Centered Kernel Alignment (CKA) [34] is a similarity index used to compare feature representations across various neural network architectures, different random initializations, or networks trained on different datasets. CKA is often applied in image classification tasks, where intermediate representations naturally capture high-level features of input images.

In our context, switching the task ID in the student network could be viewed as similar to using different networks with the same architecture, influenced by different random initializations based on task ID and trained on different datasets. However, in our policy network, the intermediate activation values do not correspond to high-level features of the input state in the same way. Here, states are numerical values, which explicitly define the relevant features. For example, if we had visual states, feature extraction would involve deriving coordinates for body parts from each state image, leading to similar extracted features across states. However, given that we work with numerical states, it is more intuitive to interpret the intermediate values as outputs of individual functions within a larger function, where the overall mapping forms a composition of these smaller functions.

Given a set of input examples, CKA first compares every pair within each representation (inner product) and then assesses the similarity structure across representations. In image classification tasks, this approach makes sense, as the network may extract similar features from images of different animals, such as dogs and cats. While the images differ, they may share common features. However, in our case, comparing different numerical states may not be meaningful, as these states may lack inherent commonalities. We recommend that future work explores how CKA can be adapted for policy networks and data in the form of numerical state-action pairs. Furthermore, it should also account for evaluation on datasets with different distributions.

Similarity Measure Invariances

Another limitation of our measurement technique is that it only compares orderings between neurons, which may not always fully reflect the shared common structure between tasks. Currently, we use Kendall's τ to compare vectors of activation values. Kendall's τ has certain invariances, such as being unaffected by monotonic transformations. This means that any transformation preserving the order of the neurons, do not affect the final similarity. This is specifically designed for ranking problems in information retrieval. However, CKA has specific invariances that are intended to capture meaningful similarities in neural network representations. Future work should explore what invariances are necessary for a similarity measure to identify common structure across tasks effectively. This should be derived from the formal definition of common structure, which is also not yet standardized.

To illustrate the complexity of defining these invariances, consider two mathematical expressions that hypothetically represent optimal behavior: (1) $x + y$, and (2) $\ln(x + y)$. Both expressions share common structure in the form of the summation, while the natural logarithm is a task-specific operation. If we modify expression (1) to $e^{\ln(x+y)}$, the computation has technically changed, but the outcome remains equivalent to the original expression, still reflecting optimal behavior. In terms of shared computation, both expressions perform a summation and apply the natural logarithm. However, in expression (1) the natural logarithm is canceled out by the exponent. This increases the observed shared computation, but without truly capturing more common structure. One possible invariance for a common structure

similarity measure could involve enforcing a minimal form of mathematical expression. However, this would be challenging to implement effectively.

Another example of a property that we believe should hold for a similarity measure in our context is that the correlation should be 1 when the activation values are exactly the same. This property holds for Kendall's τ, as it reaches a value of 1 when two sets of activation values have identical rankings, indicating perfect concordance. It is important to construct more properties like this to allow for a meaningful similarity measure with proper invariances for our context.

6.3. Computational Level Common Structure

After measuring the amount of shared computation in the current student agents, we will now try to improve it. In this section, we focus on better capturing common structure at the computational level. We first introduce an information bottleneck by reducing the network size. Afterwards, we will add a regularization term to the loss function, encouraging the student to not only mimic its teachers but also to capture common structure.

6.3.1. Reducing Network Size

The first intuition for capturing more common structure at the computational level is to reduce the size of the student network. This intuition arises from the hypothetical illustration of neuron activations across tasks (Figure 6.2) discussed in Section 6.2.1. Hypothetically, the multi-task student network could internally maintain two task-specific subnetworks. If this were the case, there would be no advantage to using a multi-task network over simply maintaining two separate single-task networks. Such a situation is only feasible if the student network has enough capacity to accommodate both tasks independently. However, if the network's capacity is reduced, it is forced to share common structure across tasks in order to preserve performance.

This approach introduces an information bottleneck [56]. An information bottleneck forces a model to capture only the essential information needed for optimal performance across all scenarios. In our case, reducing the network size restricts the student network's capacity to model both tasks separately. This forces it to merge redundant information in the common structure, while maintaining essential task-specific properties.

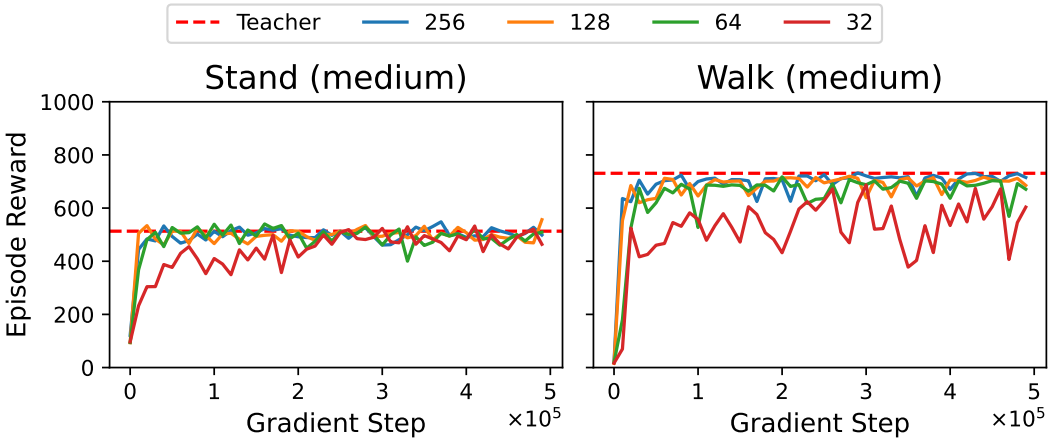
The aim of reducing the network size is to balance the performance with the shared computation. We expect that if the network is too small, there is not enough capacity to represent all essential information. This would degrade performance. If the network is too large, the information bottleneck is not fully enforced allowing the student to represent redundant information in common structure. This would decrease the amount of shared computation.

In this experiment, we vary the number of neurons in the hidden layers of the student network. This setup is similar to the experiment in Section 5.2.4. However, here we consider actual offline data instead of generated datasets with artificial distribution shifts. Additionally, we measure the degree of shared computation by computing Kendall's τ over the per-state activation values, as explained in Section 6.2.1. We use per-state measurements because it is impractical to average activation values over many states during training. The results of this experiment are presented in Figure 6.4.

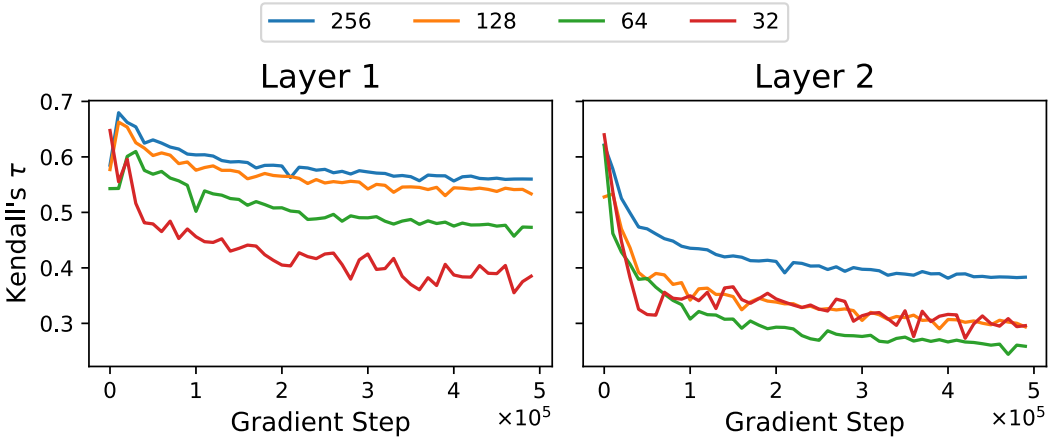
As the network size decreases, both performance and shared computation gradually degrade. This suggests that reducing network size alone does not automatically encourage sharing common structure across tasks. This is likely because it does not provide an explicit incentive to merge common behaviors across tasks. In the next section, we explore adding such an incentive directly into the loss function.

6.3.2. Ranking Regularization

We introduce a regularization term in the loss function that incentivizes the student network to achieve a high correlation in activation values across tasks. Although directly optimizing Kendall's τ would be ideal, it is non-differentiable and discontinuous, making it unsuitable as a loss function. As an alternative, we consider Normalized Discounted Cumulative Gain (NDCG) [65]. NDCG is also non-differentiable but smooth, therefore it can be used as a loss function by taking the surrogate gradient. Additionally, two other commonly used loss functions in learning-to-rank methods are the pairwise logistic loss [11] and the listwise softmax cross-entropy loss [9]. In our experiments, we evaluate all three loss functions



(a) Episode reward



(b) Per-state Kendall's τ

Figure 6.4: Results of reducing the network size evaluated on final performance (a) and shared computation (b). Each student was trained from scratch on a single random seed.

as regularization terms.

For simplicity, we define the regularized loss function in the context of our experimental setup, where the number of tasks is two. The new loss function is defined as:

$$\mathcal{L}_{regularized} := \frac{1}{2}\sum_{i=0}^{1} \mathop{\mathbb{E}}_{s\sim\mathcal{D}_i}\left[\left\|\pi_i^T(s) - \pi^S(s,i)\right\|_2^2 + \lambda\Sigma_{j=0}^{k}\mathcal{L}_{rank}\left(\ell_j(\pi^S(s,i)), \ell_j(\pi^S(s,i\oplus 1))\right)\right]$$

where the total number of tasks is 2, i is the task ID, $\mathcal{D}_i$ is the offline dataset for task i, $\pi_i^T : \mathcal{S} \to \mathcal{A}$ is the actor network of the teacher for task i, and $\pi^S : \mathcal{S} \times i \to \mathcal{A}$ is the student's actor network, λ is a hyperparameter, k is the number of layers in the student's actor network, ℓ_j retrieves the activation values of the jth layer from a network, and $\mathcal{L}_{rank}$ is the ranking loss.

The equations for the ranking losses are as follows:

- Normalized Discounted Cumulative Gain (NDCG):

$$\text{NDCG}(x,y) := -\frac{\sum_{i=1}^{n}\frac{y_{\pi_x(i)}}{\log_2(i+1)}}{\sum_{i=1}^{n}\frac{y_{\text{sorted}(i)}}{\log_2(i+1)}},$$

  where $\pi_x(i)$ is the permutation of item i induced by x.

- Pairwise logistic loss:

$$\mathcal{L}_{pair}(x,y) := \sum_{i=0}^{n}\sum_{j=0}^{n}\log\left(1 + e^{-(x_i-x_j)\cdot(y_i-y_j)}\right)$$

- Listwise softmax cross-entropy loss:

$$\mathcal{L}_{list}(x,y) := -\sum_{i=1}^{n} P_y(i)\log P_x(i),$$

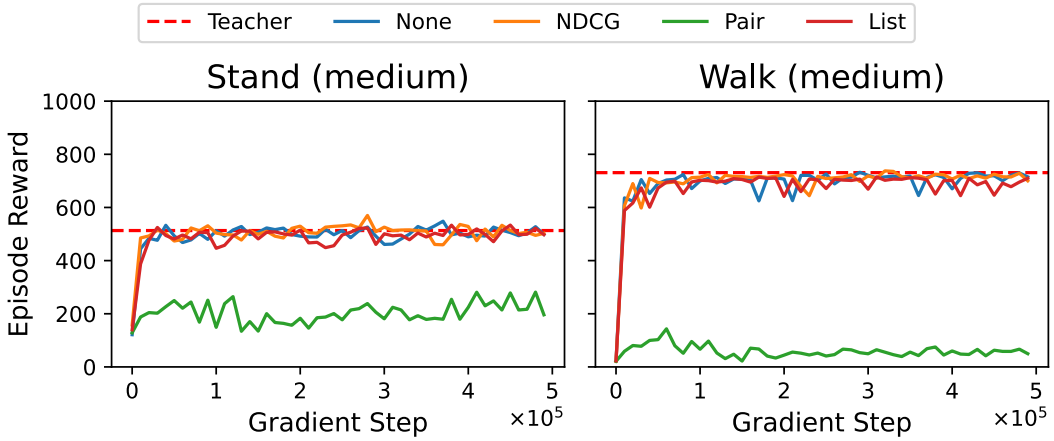
  where $P_z(i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$.

Each ranking loss takes two vectors of ranking scores and computes a similarity in terms of ordering. In our context, these two vectors correspond to the neuron activation values of a layer in the standing task and the walking task. Note that this is a per-state comparison of activation values, as described in Section 6.2.1. We use per-state comparisons because averaging activation values over all states in the offline data is impractical for every gradient step. By minimizing these losses, we encourage the same ranking of neurons across tasks, effectively guiding the student network to activate the same neurons irrespective of the task ID.

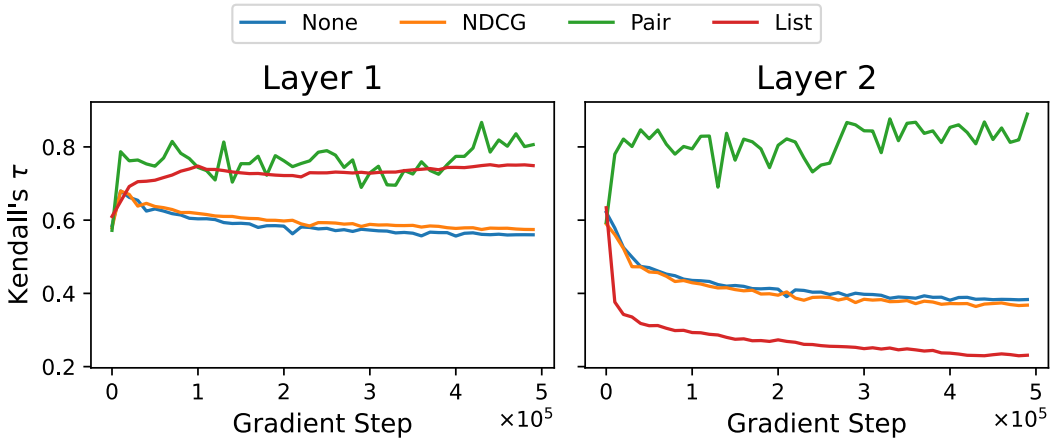
This regularization term serves as an additional objective to the primary objective of replicating the teachers' actions. The hyperparameter λ controls the relative importance of this objective. When λ is set to 0, the optimization focuses only on replicating the teachers' actions. In contrast, a high λ value leads the student to prioritize maximizing shared computation. So we are essentially balancing between the two objectives with the goal to deviate slightly from the teacher actions where it can significantly increase shared computation. To reduce computational overhead, we did not perform extensive hyperparameter tuning but found that setting λ to 100 best illustrates the effect in our experiments.

The results are displayed in Figure 6.5. The pairwise loss achieves the highest Kendall's τ, but this comes at a significant cost to performance. This suggests that the student effectively learns to ignore the task ID. However, the primary objective of replicating the teachers' actions is overshadowed, leading to a degradation in performance. This outcome indicates that negative transfer occurs, as the student network is encouraged to increase shared computation across tasks, resulting in suboptimal actions for both tasks.

The NDCG loss shows results almost identical to those with no regularization in both episode reward and Kendall's τ. This is likely due to the difficulty of optimizing the surrogate gradient. Consequently,



(a) Episode reward



(b) Per-state Kendall's τ

Figure 6.5: The results of the ranking regularization experiment with $\lambda = 100$ evaluated on performance (a) and shared computation (b) where each line represents a different type of ranking loss. Each student was trained from scratch on a single random seed.

the optimization process may fail to meaningfully impact the model's behavior, resulting in little difference compared to no regularization.

Lastly, the listwise loss maintains the same performance while significantly improving Kendall's τ in the first layer. However, in the second layer, the shared computation decreases, averaging out to a level similar to that with no regularization. This suggests that shared computation can be enforced across tasks in one layer, but this transformation may be undone in a deeper layer. This behavior is the same as in the example of mathematical expressions discussed in Section 6.2.4. We considered two expressions: (1) $x + y$ and (2) $\ln(x + y)$. The shared computation can be increased by modifying expression (1) to $e^{\ln(x+y)}$. In this way, the natural logarithm could be applied in one layer to increase shared computation, but be canceled out in a deeper layer by raising the exponent. On average, this process hypothetically maintains the same level of shared computation.

In conclusion, adding a regularization term to increase shared computation in the loss function does not improve performance or increase shared computation. The main challenge with ranking regularization lies in the abstract nature of shared computation. As discussed in Section 6.2.4, our technique for measuring shared computation has limitations in fully reflecting the captured common structure. While it can be somewhat useful as a measurement tool, directly optimizing this measure is less intuitive. This is because the measure does not fully represent the goal of capturing more common structure. Without a clearly defined objective for shared computation, optimization becomes difficult. This makes it challenging to ensure that the desired type of behavior is shared.

6.4. Trajectory Level Common Structure

In this section, we focus on capturing common structure at the trajectory level. We begin by discussing the fundamental capabilities of PD and conclude that PD can only merge common structure in states where the teacher behaviors are aligned. We argue that multi-task exploration is necessary to align teacher behaviors. To enable multi-task exploration, we introduce the assumption that the reward functions are known. Based on this assumption, we propose two extensions to our approach aimed at aligning teacher behaviors through multi-task exploration: Data Sharing (DS) and Offline Q-Switch. Finally, we discuss the limitations of these extensions.

6.4.1. Multi-Task Exploration

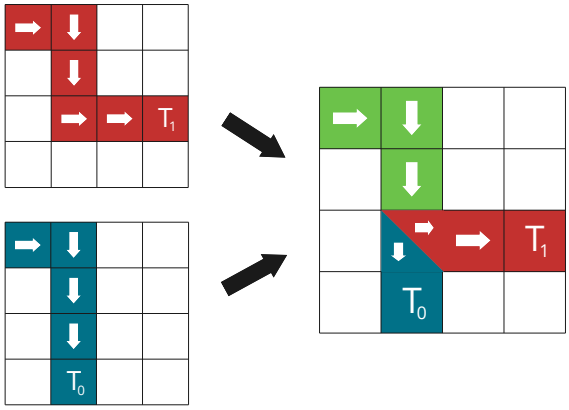
In Chapter 5, we demonstrated that MOP can successfully learn a multi-task student network that matches the performance of its teachers. However, in Section 6.2, we observed that the extent of shared computation across tasks within the student network is limited. This raises a fundamental question: why does the student fail to leverage common structure to outperform its teachers? PD essentially copies the behavior of its teachers, merging common behaviors and separating different behaviors. This approach is effective at avoiding negative transfer, but it can miss common structure if the teachers do not behave similarly in those parts of the tasks. In other words, common structure at the trajectory level is not automatically captured by PD if the teacher behaviors are unaligned.

There are two situations in which the behaviors of the teachers are unaligned: 1) at least one teacher is non-expert, or 2) alternative behaviors are equally optimal. To explain this, we will use the grid world example from Chapter 6.1.

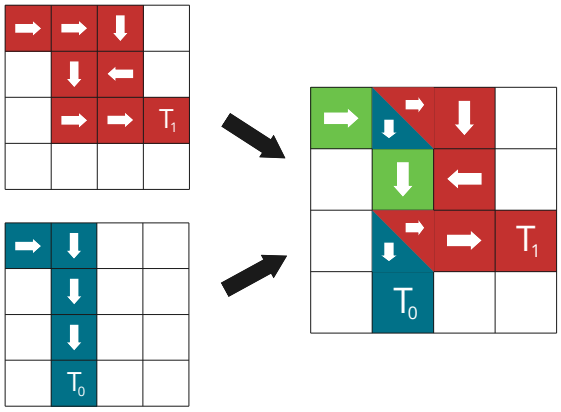
In the ideal case, the teachers follow an optimal path from the top left to their target and they follow the same trajectory as much as possible. PD will merge the state-action pairs that are task independent. This is shown in Figure 6.6a, where three state-action pairs are shared across tasks which is optimal for the declared targets.

If at least one teacher is non-expert, it is likely to take a different action in a common structure state than the other teacher. This is visualized in Figure 6.6b. The red agent, aiming to reach T_1, takes a longer path than necessary. However, the distillation process does not correct for this. It simply copies the behavior of the teachers and identifies the detour as a task-specific property. This specific example results in one state-action pair less being shared across tasks, than the ideal scenario.

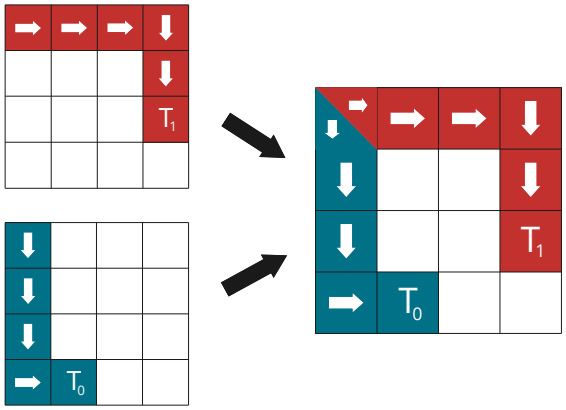
If the teachers are optimal but equally optimal behaviors exist, the behaviors can be unaligned. This is illustrated in Figure 6.6c. Both teachers follow an optimal path to their target, but there are no



(a) Optimal and aligned



(b) Suboptimal



(c) Unaligned

Figure 6.6: Three scenario's of single-task teachers (left) distilled into a multi-task student (right) in a grid world. The trajectories of task 0 are in blue, of task 1 in red, and the common structure is green.

overlapping parts of the trajectories. When distilling from these teachers, PD will identify all state-action pairs as task-specific. Thus no common structure is captured in this example.

These scenario's show that the amount of common structure captured by PD is highly dependent on the behaviors of the teachers. It will not merge state-action pairs which are not already the same across teachers. This reasoning leads to Hypothesis 2.

Hypothesis 2 *Regular Policy Distillation can only share common structure in states where the teachers behave identically.*

A consequence of Hypothesis 2 is that the student cannot outperform its teachers in regular PD. This occurs because common behaviors in the teachers are merged, while different behaviors are separated. There is no incentive for the student to compare different behaviors in the same state and merge them by selecting the best performing one. The reason for this limitation is the absence of multi-task exploration; the student has no way of knowing how effective a teacher's behavior would be in the other task. Multi-task exploration would allow to try behaviors of teachers in another task. If multi-task exploration shows that one behavior yields greater or equal reward across all tasks, the student can copy it and treat it as common structure. If each behavior performs best only for its own task, it should be preserved as task-specific.

The most straightforward way to introduce multi-task exploration would be to test the behavior in the environment and observe the resulting reward. Distral [61] is a method that incorporates this by alternately distilling knowledge from single-task policies into a multi-task policy and regularizing the single-task policies with the multi-task policy. This regularization promotes exploration of behaviors learned in other tasks. However, in the offline setting, online exploration is not possible.

Another intuitive approach is to use the teachers' Q-values. Q-switch Mixture of Policies (QMP) [80] is a method which selects the best action across all single-task policies at a given state by evaluating it in the Q-value function of the current task. QMP operates online and adapts single-task Q-value functions by observing the reward while performing multi-task exploration. However, offline RL algorithms typically learn pessimistic value functions, penalizing state-action pairs outside the training distribution to avoid catastrophic behavior upon deployment. As a result, evaluating a state-action pair from one task using the Q-value function of another would yield a low value if the pair is outside the training distribution. If the pair is inside the distribution, it would yield a high value, but in that case the teacher would have already learned the best action for that state.

To enable multi-task exploration in our setup, we assume that the reward functions are known. We formalize this in Assumption 1. In the following sections, we explore two approaches for leveraging these reward functions to better capture common structure: Data Sharing (DS) and Offline Q-Switch (OQS).

Assumption 1 *The reward functions of all tasks, $\{R_i\}_{i=0}^{m-1}$, are known to the agent, either provided directly or accurately estimated.*

6.4.2. Data Sharing

As discussed in Chapter 3, prior work has demonstrated that sharing offline data from related tasks can enhance single-task policies [76, 4, 75]. This concept can be incorporated into MOP by relabeling the data with the known reward functions and adding it to each task's offline dataset. This adaptation is illustrated in Figure 6.7. For each task, the transitions in all offline datasets are relabeled by applying the transition (s, a, r_j, s') to the reward function of the target task: $r_{j\to i} = R_i(s, a, s')$, where R_i is the reward function for task i, s is the initial state, a is the action taken, and s' is the resulting state. The mixed offline datasets, containing all relabeled transitions, are then used by the offline RL algorithm to learn the teacher policies. The distillation process remains the same as in MOP, with the only difference being that states are now sampled from the mixed datasets. In the original approach, the states sampled for distillation were separated by task, whereas in this approach, all states from all tasks are fed to all teacher policies.

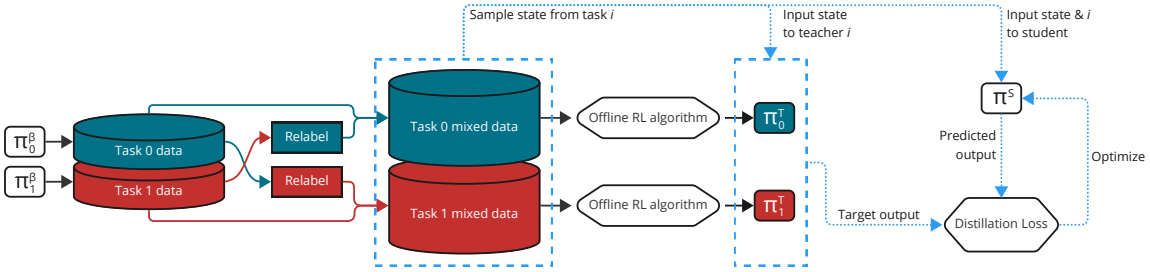


Figure 6.7: Data sharing incorporated into MOP. Task 0 is marked in blue and Task 1 in red to highlight the relabeling of data.

The adapted distillation loss becomes:

$$\mathcal{L}_{DS} := \frac{1}{m} \sum_{i=0}^{m} \mathop{\mathbb{E}}_{s \sim \mathcal{D}} \left[\left\| \pi_i^{T_{DS}}(s) - \pi^S(s, i) \right\|_2^2 \right],$$

where m is the total number of tasks, i is the task ID, $\mathcal{D} := \bigcup_{i=0}^{m} \mathcal{D}_i$ is the union of all task-specific offline datasets, $\pi_i^{T_{DS}}$ is the actor of the teacher for task i trained on relabeled data $\bigcup_{j=0}^{m} D_{j \to i}$, and π^S is the student's actor.

However, DS can also exacerbate distribution shift and degrade offline RL performance [76]. This occurs because the offline data is collected by different behavior policies for each task, leading to significantly different state distributions. The final state distribution of the mixed dataset is likely much broader and may contain many low-reward transitions for the target task, as these were collected under different objectives. Offline RL algorithms aim to avoid low-reward state-action pairs, so the teacher policy is likely to follow a much narrower state distribution. This can exacerbate the distribution shift relative to the original offline dataset. In particular, offline RL algorithms that constrain the teacher policy to stay close to the behavior policy are vulnerable to performance degradation due to large distribution shifts. In contrast, uncertainty-based algorithms tend to be more robust to this issue [4]. We are using the uncertainty-based PBRL algorithm [5], which helps mitigate the performance degradation caused by a distribution shift.

In the context of PD, DS serves as a form of multi-task exploration, enabling trajectories collected by behavior policies from all tasks to be cross-evaluated and included in each offline dataset. This approach should help the offline RL algorithm learn better teacher policies if DS improves the coverage of optimal trajectories, especially in cases where a teacher would otherwise be non-expert. Ideally, DS results in expert teachers across tasks, thereby avoiding the issue of unmergable teacher behaviors caused by suboptimal actions, as discussed in Chapter 6.4.1. While it does not fully address the challenge of equally optimal but unaligned teacher behaviors in common structure states, we are working with continuous state and action spaces, making it highly unlikely that behaviors will be exactly equally optimal. DS ensures that teacher policies are learned on the same distribution of state-action pairs. Given the continuous nature of the state and action spaces, we expect that the teacher policies will naturally align if they can learn expert behavior. Having aligned teacher behaviors across common structure states would allow PD to merge these behaviors and fully leverage the common structure.

6.4.3. Offline Q-Switch

As mentioned earlier, DS can exacerbate distribution shift, leading to a degradation in offline RL performance. While an uncertainty-based offline RL algorithm can mitigate this, the risk is not entirely eliminated. One potential solution to this issue is sharing policies instead of data. QMP [80] is a method that implements this approach in the online setting. It compares the actions of all task-specific policies and selects the highest-valued action according to the Q-value function of the target task. This action is then executed in the environment, and the policy and Q-value function of the target task are updated accordingly.

While QMP provides an effective approach to multi-task exploration, it is not feasible in the offline

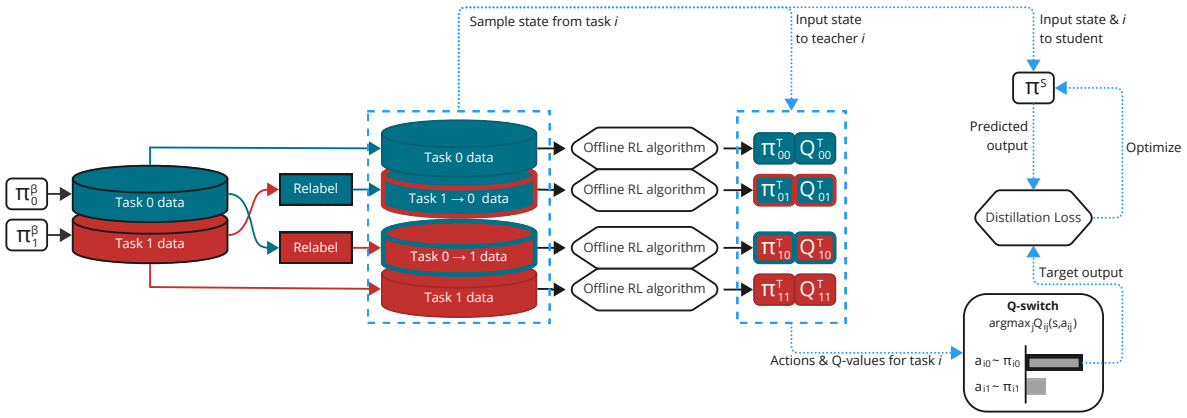


Figure 6.8: Q-switch incorporated into MOP. Task 0 is marked in blue and Task 1 in red. The relabeled datasets are circled with their original colors to emphasize that the transitions still follow the distribution of the original offline datasets, while the rewards have been relabeled to the target task.

setting. Offline RL algorithms typically learn pessimistic Q-value functions, assigning low values to state-action pairs that are out-of-distribution relative to the offline data. Evaluating a state-action pair from another task using such a pessimistic Q-value function will result in a low value if the pair is out-of-distribution, offering no contribution to exploration. State-action pairs within the offline data distribution might yield high values, but these are likely already known by the policy as good actions.

Nevertheless, since the reward function is known, we can instead learn multiple critics that accurately evaluate each task's state-action pairs in another task. Rather than relabeling the data and merging it into a single dataset as in DS, we can maintain separate datasets for each relabeled set and learn individual teachers from them. This approach also avoids exacerbating the state distribution shift. The highest-valued action can then be used as the target output for distillation. We refer to this technique as Offline Q-Switch (OQS).

The extension of MOP with OQS is shown in Figure 6.8. For each task, the transitions from all offline datasets are relabeled according to the target task. However, these relabeled datasets remain separate. The offline RL algorithm learns a teacher from each dataset, retaining both the actor (policy) and the critic (Q-value function). During distillation, a state is sampled, and all teachers for the target task select an action and evaluate it using their respective critics. The Q-switch mechanism selects the highest-valued action as the target output for the student to mimic.

The new loss function is defined by:

$$\mathcal{L}_{OQS} := \frac{1}{m}\sum_{i=0}^{m} \mathop{\mathbb{E}}_{s\sim\mathcal{D}} \left[\left\| \underset{0\le j<m}{argmax} \left(Q_{ij}^{T_{OQS}}(s, \pi_{ij}^{T_{OQS}}(s)) \right) - \pi^S(s,i) \right\|_2^2 \right],$$

where m is the total number of tasks, i is the task ID, $\mathcal{D} := \bigcup_{i=0}^{m} \mathcal{D}_i$ is the union of all task-specific offline datasets, $Q_{ij}^{T_{OQS}}$ and $\pi_{ij}^{T_{OQS}}$ are the critic and actor of the teacher for task i trained on relabeled data $D_{j\to i}$, and π^S is the student's actor.

This approach will significantly increase training time, as it requires the offline RL algorithm to run m^2 times instead of m. However, it has the potential to improve performance and reduce the required network capacity. This is because the original teacher, trained on the original offline dataset, is included among the teachers trained on relabeled data. The idea is that this teacher serves as a baseline, and its actions are only overridden if a higher-valued action is found from one of the other teachers.

6.4.4. Results

To test the DS and OQS extensions, we evaluate three objectives: teacher performance, student performance, and the amount of shared computation across tasks.

Data Sharing

We expect DS to improve teacher performance in an absolute sense, as it is trained on twice as much data. It is important to note that the sample complexity remains unchanged; we are not collecting additional data but rather utilizing the available offline data more efficiently by sharing it across tasks. However, performance may degrade if sharing significantly exacerbates the distribution shift.

We anticipate that teacher behavior in common structure states will naturally align due to DS. The teachers are trained on the same offline data, with the only difference being the relabeled rewards. In common structure states, the optimal action is the same across tasks. Since all teachers have access to the same data, they are likely to converge on the same action in these states. This does not address the issue of equally optimal behaviors, where multiple actions may yield similar value in common structure states. Without communication between the teachers during training, unaligned behavior could still occur. However, given the continuous state and action spaces, it is highly unlikely that different behaviors would result in exactly the same reward. Therefore, we expect that if the offline RL algorithm is capable of learning high-performing teachers, their behaviors will naturally align.

If the teachers show improved performance and aligned behavior, we expect student performance and the amount of shared computation to improve as well. This expectation follows directly from Hypothesis 2. Teachers with identical behavior can be effectively merged by PD, leading to increased shared computation. Since we anticipate that teacher performance will improve and the student typically matches this performance, the student's performance is also expected to improve.

The results indicate that DS is particularly effective in the Medium X Medium setting (Figure 6.9a). The teacher improves upon MOP in the standing task and matches its performance in the walking task. The student successfully replicates this behavior, achieving the best performance compared to both MOP and OQS across both tasks. Additionally, shared computation improves significantly compared to MOP (Table 6.2), which aligns with our expectations.

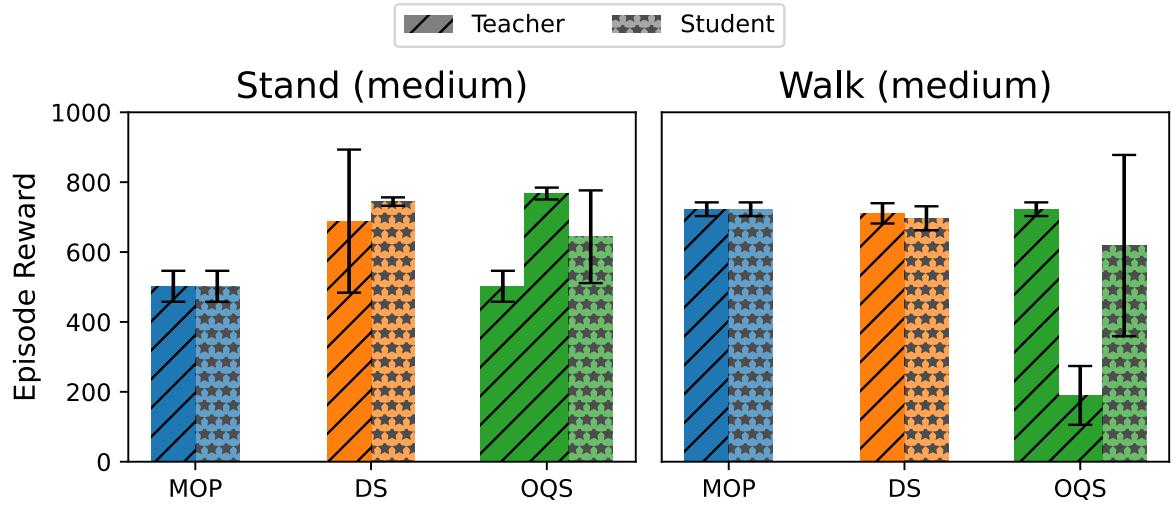
However, in the Expert X Expert setting (Figure 6.9b), DS significantly degrades teacher performance compared to MOP in both tasks. This is likely due to an increased distribution shift. The expert datasets consist of high-value trajectories specific to the target task. When data is shared across tasks, the newly introduced trajectories are likely to have lower value, as they were collected by expert behavior policies from another task. While this increases the overall coverage of state-action pairs, it relatively reduces the proportion of optimal trajectories. Therefore, we conclude that DS can degrade performance when the behavior policies are already highly performant. As a result, the students learned from these teachers also perform poorly. Nonetheless, the shared computation (Table 6.2) remains significantly higher than in MOP, which can be attributed to the teachers being trained on the same mixed data.

Offline Q-Switch

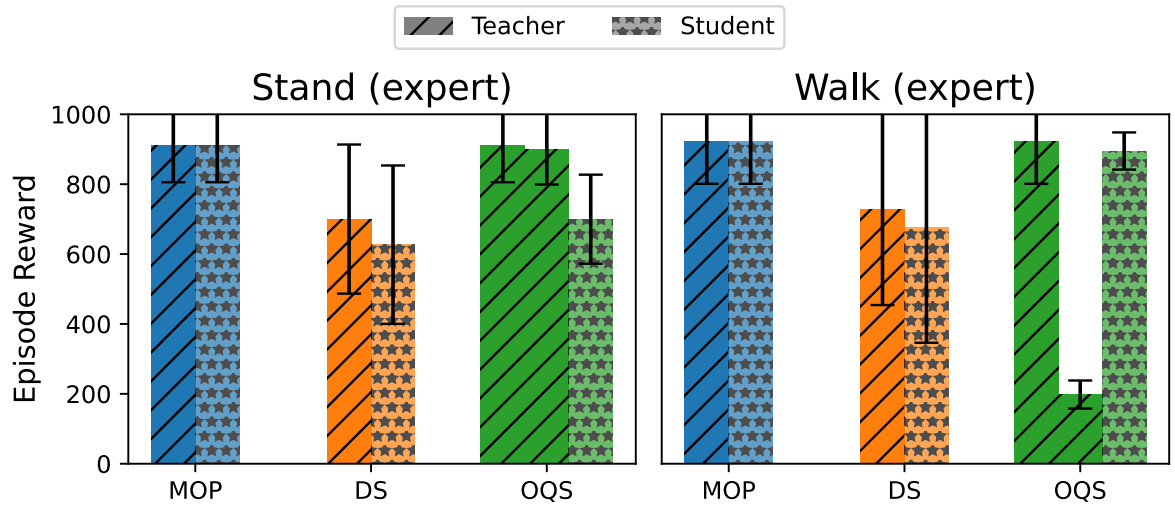
Aside from DS, OQS is not always expected to improve teacher performance. With OQS, we learn two teachers per task: one trained on the original offline data of the target task (as in MOP), and one trained on the relabeled data from the other task. Ideally, the Q-switch ensures that the original teacher is only overruled by the cross-task teacher if it suggests a higher-valued action. Therefore, we expect the OQS student to perform at least as well as the MOP student.

We also expect OQS to increase shared computation compared to MOP. Unlike DS, OQS does not train all teacher policies on the same mixed data. Instead, it keeps the relabeled data separate, preventing an exacerbation of the distribution shift. As a result, the teacher policies are not expected to show fully aligned behavior. However, the student mimics the best action from either the original or cross-task teacher for each task. This should result in the student selecting the same actions in common structure states across tasks. In other words, while the teachers are not trained on the mixed datasets, the student indirectly leverages data from both tasks via the Q-switch mechanism. Consequently, we expect that behaviors will align in common structure states, allowing the student to merge them effectively.

The results shown in Figure 6.9 present two teachers for OQS. The left bar corresponds to the teacher



(a) Medium X Medium



(b) Expert X Expert

Figure 6.9: Results for MOP, extended with Data Sharing (DS) and Offline Q-Switch (OQS). Note that OQS maintains two teachers per task: the left bar corresponds to MOP's teacher ($\pi_{ii}^{T_{OQS}}$), and the right bar represents the teacher trained on relabeled data from the other task ($\pi_{ij}^{T_{OQS}}$). The experiments were conducted using 4 random seeds for each bar. Error bars represent the 95% confidence intervals.

Student (Walk X Stand)	Approach	τ Layer 1	τ Layer 2
Medium X Medium	MOP	0.067	0.15
	DS	0.52	0.67
	OQS	0.68	0.76
Expert X Expert	MOP	0.16	0.36
	DS	0.44	0.62
	OQS	0.43	0.72

Table 6.2: A comparison of Kendall's τ across different student models, training approaches, and layers.

trained on the original data ($\pi_{ii}^{T_{OQS}}$, where i is the target task), which is the same as the MOP teacher. The right bar represents the cross-task teacher ($\pi_{ij}^{T_{OQS}}$, where i is the target task and $j \neq i$), trained on the relabeled data from the other task.

In the standing task, the cross-task teachers outperform or match the original teachers (the right bar is higher than the left bar), suggesting that using walking data to learn to stand is effective. This makes sense because the walking data contains many instances of standing, which can be leveraged to learn the standing task. The state-action pairs related to walking can be disregarded when learning to stand. However, in the walking task, the cross-task teachers perform poorly (the right bar is lower than the left bar). This is expected, as the standing data contains no examples of walking behavior. It can only learn how to stand, leading to lower rewards under the reward function of walk.

The key idea of Q-Switch is that the student replicates the best-performing teacher for any given state. Therefore, the student is expected to match the performance of the best of the two teachers across all tasks. Contrary to these expectations, the OQS student always performs worse than its best teacher. We suspect this is due to the lack of proper coordination between the different Q-values being maximized. Switching between different teachers may increase the per-action Q-value, but it could disrupt the overall reward obtained over the entire trajectory.

In terms of shared computation, as shown in Table 6.2, OQS performs the best. However, since the reward decreases, some of this sharing may be attributed to negative transfer.

6.4.5. Limitations

In the previous experiment, it was shown that data sharing can produce high-performing teachers, especially in the stand task when trained on the medium offline dataset (Figure 6.9a). However, this approach significantly degrades performance when applied to the expert offline datasets (Figure 6.9b). Data sharing has been extensively explored in prior work, with significant claims about its performance. However, we also want to point out the limitations of the technique.

We found that the high performance of data sharing can be attributed to an unintended feature in the environment. To demonstrate this, we deployed each teacher to solve the other task: the walk teacher was tested with the stand reward function, and vice versa ($R_{i \oplus 1}$). The results, shown in Table 6.3, indicate that the walk teacher performs better in both tasks for all offline datasets. This suggests that simply copying the walk teacher could maximize rewards across tasks, rather than learning a multi-task policy. In other words, there is no negative transfer from the walk task to the stand task. To prevent this unintended feature, task-specific properties should be added to the stand task, such as a negative reward for movement.

Teacher	R_{stand}	R_{walk}
Walk expert	**904**	**862**
Stand expert	898	204
Walk medium	**752**	**729**
Stand medium	514	181
Walk medium-replay	**669**	**623**
Stand medium-replay	503	139

Table 6.3: A comparison of teachers performing each other's task shows that consistently using the walk teacher yields the highest reward.

Upon reviewing the Uncertainty-driven Data Sharing (UTDS) method [4], we hypothesize that its success depends on the incremental structure of the tasks in their experiments. It would be interesting for future work to explore the effectiveness of this method in an environment where negative transfer occurs between all tasks, forcing it to operate in a true multi-task setting.

Our work does not rely on this environmental unintended feature. Although the stand reward function does not enforce stationarity, our student agents did learn to remain stationary (except under

the data sharing extension). We also explored using a new environment with clearly disjoint task-specific properties. However, the tasks proved too challenging for TD3 and PBRL to solve within our computational constraints. Refining this computationally intensive experimental setup is beyond the scope of this work. A better solution to address the unintended feature would be to modify the DeepMind Walker environment by introducing a negative reward for forward movement in the stand task, which we leave for future work.

6.5. Discussion

In this chapter, we aimed to capture the common structure between tasks through PD. In Section 6.1, we defined common structure as behaviors that can be shared across tasks at either the trajectory or computational level. However, defining common structure is challenging due to the abstract nature of behaviors and their dependency on the multi-tasking method. Future work should focus on standardizing the definition of common structure. This could help comparing and improving methods aimed at capturing it across tasks.

To the best of our knowledge, we proposed the first method to quantify common structure shared across tasks. Specifically, we measured shared computation as an indicator of the extent of common structure captured across tasks. We analyzed the correlation of neuron activations on a per-state basis as well as averaged across all states encountered during task performance. Each approach has its limitations: per-state comparisons do not account for the tasks' state distributions, while averaging activations can cause them to fade out against one another. To measure the similarity between activation values across tasks, we employed Kendall's τ, which intuitively ranks neurons based on their consistent activation for each task and compares these rankings across tasks. Although Kendall's τ provides a useful indication of the level of shared structure, it may not capture all aspects of a formal definition of common structure. We recommend that future work addresses the limitations of comparing activation values with respect to the tasks' state distributions and establishes invariance properties of a similarity measure that aligns with a formal definition of common structure.

The results of measuring the shared computation of the students from Chapter 5, showed that tasks trained on medium offline datasets shared very little computation. Nonetheless, those trained on expert datasets shared significantly more. This suggests that better performing teachers are more likely to share behaviors in common structure. We suspect this is because the teachers learned optimal behaviors, which tend to be more aligned across tasks.

To further enhance PD's ability to capture common structure at the computational level, we reduced the network size and introduced ranking regularization. Reducing the network size alone did not enhance shared computation, due to the absence of an explicit incentive. Ranking regularization was designed to provide this incentive by adding a regularization term to the loss function. This encourages the student network to align neuron activation values more closely across tasks. Using a pairwise logistic loss improved shared computation, but degraded performance. The listwise softmax cross-entropy and Normalized Cumulative Gain (NDCG) losses did maintain performance, but the shared computation remained constant on average. The main challenge with ranking regularization is the abstract nature of shared computation: it is not entirely clear what shared computation should look like to best capture common structure. The lack of a well-defined goal for shared computation complicates optimization, making it difficult to determine whether the correct kind of shared behavior is being encouraged.

When focusing on common structure at the trajectory kevel, we found that a key limitation of regular PD, as shown by Hypothesis 2, is that it can only merge behaviors where the teachers are identical. If teachers show different behaviors for the same state, PD treats these behaviors as task-specific properties, keeping them separate in the student.

To address limitation, teacher behaviors must be aligned. Multi-task exploration could help facilitate this by evaluating behaviors from one task in the context of another. We proposed two extensions that introduce a form of multi-task exploration: Data Sharing (DS) and Offline Q-Switch (OQS). Both of these extensions require the reward functions to be known. DS relabels offline data from one task with the reward function of the target task and shares all data to train the teacher policies. The results indicated that DS can improve teacher and student performance when data from related tasks is shared. It enhances shared computation across tasks as well. However, DS can also degrade performance due to

an exacerbated state distribution shift. OQS, on the other hand, selects the best action between two teachers trained on separate relabeled datasets. While it showed improved shared computation, the overall student performance decreased. This is probably due to a lack of coordination between the teachers' Q-values, leading to suboptimal decisions over an entire trajectory.

For future work, we recommend investigating alternative methods for multi-task exploration. Skills Regularized Task Decomposition (SRTD) [74] is a related method focused on offline MTRL. As discussed in Section 3.2, SRTD augments offline data by sampling imaginary trajectories from learned skill and task embeddings. The authors suggest that these imaginary trajectories are similar to expert demonstrations. This presents an appealing approach for multi-task exploration, because skills shared across tasks can be leveraged to generate trajectories with aligned behavior. However, it is unclear from the SRTD results whether this alignment is effectively achieved. The authors do not compare their method to single-task offline RL approaches, leaving it uncertain whether there is a performance gain from exploiting common structure across tasks. It is possible that the latent space does not accurately represent shared skills, as it is trained on limited offline data. As a result, the generated trajectories might still fail to align across tasks, thereby preventing effective multi-task exploration.

It would be interesting to investigate whether incorporating context through Large Language Models (LLMs) [1] could enable more meaningful exploration across tasks. Recent research has demonstrated that the broader contextual understanding of LLMs can be used to generate high-quality imaginary trajectories for exploration in offline RL. Knowledgeable Agents from Language Model Rollouts (KALM) [47] is a technique that leverages natural language descriptions of skills along with corresponding offline data to produce diverse rollouts that reflect new skills. Results show that KALM is capable of effectively executing tasks with previously unseen goals. While this approach requires textual context to be available for each task, it has the potential to generate trajectories with aligned behaviors across tasks. Therefore, it could serve as a useful extension to our approach.

Lastly, our experiments revealed limitations in the environment. We found that deploying the walking teacher in the stand task consistently outperforms the standing teacher at its own task. Rather than learning a proper multi-task student, it is more effective to simply copy the walking teacher and deploy it for both tasks. While our work did not exploit this unintended feature, introducing a penalty for movement in the stand task would help prevent this issue. We leave this improvement for future work.

In conclusion, while PD successfully avoids negative transfer, it can fail to capture common structure when teacher behaviors are unaligned. We explored several approaches to address this: ranking regularization, DS, and OQS. Ranking regularization did not improve performance, as the objective is too abstract. Both DS and OQS introduced forms of multi-task exploration, which showed promise in certain scenarios but also revealed clear limitations in others. Future research should focus on standardizing definitions of common structure, developing more effective multi-task exploration methods for the offline setting, and designing environments that better reflect the actual multi-task setting.

7

General Discussion

This chapter presents a general discussion on the limitations, future work, and broader topics related to our work. We emphasize the relevance of offline MTRL, discuss the general limitations of PD, and address the limitations of our experimental setup.

7.1. Relevance of Offline Multi-Task Reinforcement Learning

This work addressed the offline MTRL setting. In the introduction, we briefly outlined its relevance, but we would like to emphasize this again. We believe that both the offline and multi-task settings are essential for applying RL to real-world problems. The offline setting enables safe deployment by relying on previously collected data, but it may be constrained by the coverage and quality of the available offline data. On the other hand, MTRL contributes to general artificial intelligence by combining skills across tasks, which is particularly valuable in the offline setting. MTRL can use offline data from different tasks, allowing RL agents to maximize the use of available samples. In this section, we emphasize the relevance of offline MTRL from three perspectives. First, how it contributes to key open problems in RL. Second, the importance of capturing common structure rather than simply scaling up resources. Third, the potential of offline MTRL to address real-world problems effectively.

Relevance to Open Problems

As discussed in previous work, two major open problems in RL are generalizability and deployability [45]. Generalizability refers to the agent's ability to perform well across diverse tasks and environments, while deployability focuses on training agents in a way that prevents catastrophic actions in the real world.

Offline MTRL addresses both of these challenges when an agent successfully captures common structure across tasks. MTRL enhances generalizability by enabling the agent to learn multiple tasks simultaneously, allowing shared behaviors to improve performance across tasks. Offline RL enhances deployability by training agents only on previously collected data, thus avoiding the risk of catastrophic behavior that can arise from interactions with the environment during training.

Relevance of Capturing Common Structure

We mentioned that it is crucial to capture common structure to be successful in MTRL. However, in practice, it is also possible to scale up resources and maintain redundant single-task policies without merging common behaviors, achieving high accuracy across tasks. While this approach may work in the short term, it does not reflect true multi-tasking and has significant scalability challenges, especially when dealing with a large number of tasks.

This is similar to the generalization problem in supervised learning, where spurious correlations in the training data can be exploited to achieve high accuracy [2]. For instance, in a classification task, a model might rely on irrelevant features (such as the background in an image) to predict the label correctly, which results in overfitting. Although this could yield high performance in the training domain,

the model's accuracy degrades when exposed to a new domain where those spurious correlations are no longer present. To ensure out-of-distribution generalization, it is necessary to learn invariant relationships that are consistent across different environments. Similarly, in MTRL, merging shared behaviors across tasks is essential for capturing common structure. When dealing with hundreds of related tasks, this approach reduces the amount of resources significantly and can enhance performance across all tasks. An intuitive analogy is a set of socket wrenches, each designed for a specific bolt size. It is much more efficient to have a single adjustable socket wrench that can adapt to any bolt size.

Relevance to Real-World Problems

As an example of the practical relevance of MTRL, prior work shows how robotic arms can learn to grasp a wide range of objects [31]. In this setup, rather than learning separate grasping policies for individual objects, the robot arms learn a single, adaptable policy capable of generalizing to previously unseen objects. This generalization is enabled by leveraging a large, diverse offline dataset and focusing on skills that apply across multiple objects. Thereby it enhances grasping efficiency and adaptability without requiring extensive new data for each individual object. It is worth noting, however, that this approach is not fully offline. While it starts with an extensive initial dataset, the robot arm gathers additional data during training through interaction with the environment. This allows the model to explore new trajectories, which is not possible in domains where policy exploration is dangerous, such as in healthcare. Therefore, it remains important to address the truly offline setting, where no interaction with the environment is allowed before deployment.

Another example is the development of self-driving cars, which require extensive offline data to operate reliably in real-world scenarios. Large datasets from diverse driving environments enable the car's policy to generalize and perform complex tasks, such as urban driving [52] or highway driving [42]. The reward functions for urban and highway driving share components for lane following and speed control. Additionally, both tasks can benefit from sharing behaviors, such as detecting other vehicles. However, there are also task-specific properties. For example, urban driving requires responsiveness to pedestrians crossing the road, a scenario that does not arise on the highway. Applying offline MTRL techniques can maximize the value of the previously collected data by learning a single policy that can handle both tasks. This approach could improve performance across both settings while requiring less capacity in the car's self-driving system.

7.2. Limitations of Policy Distillation

While PD is a powerful technique for capturing common structure and avoiding negative transfer, it comes with several limitations. One of the drawbacks is the training time. PD requires training single-task agents before distilling their behaviors into a student network. This two-stage process can be time consuming, especially when dealing with a large number of tasks. However, in the offline RL setting, training time is generally a lower priority [38]. The focus tends to be more on maximizing the use of available data rather than minimizing computation time. Therefore, the increased training time is less critical in the offline setting.

As previously discussed, a major limitation of PD in the offline setting is its inability to perform multi-task exploration during training, unlike methods such as Distral [61] and QMP [80]. In the online setting, these methods enable the agent to evaluate behaviors from one task in the context of another which enhances multi-task learning. However, in the offline context, exploration is not possible because the agent is restricted to learning from a fixed offline dataset. This means PD can only rely on the state-action pairs present in the previously collected data. As explained in Section 6.4.1, PD is limited to capturing common structure in states where the teachers show identical behaviors, making it essential for teacher behaviors to be aligned. Without multi-task exploration, achieving this alignment is highly challenging. While we have attempted to introduce forms of multi-task exploration by assuming the reward functions are known, this approach has proven to be difficult. We recommend that future work explores other ways to enable multi-task exploration in the offline setting to address these limitations.

7.3. Limitations of Experimental Setup

While our experimental setup provides useful insights into PD in the offline MTRL setting, several limitations remain that could be addressed in future work.

The environment we used involves large continuous state and action spaces, which make it difficult to measure state-action distributions and clearly define common structure. However, using smaller, discrete spaces would mitigate offline RL challenges. So our choice of environment reflects the practical challenges in offline RL, but it also introduces difficulties in analysis.

Our experiments focus on only two tasks: standing and walking. While this is enough for investigating shared behavior, we did not test the performance when learning more tasks. Future work could explore a larger set of tasks to further test the effectiveness of our approach. Additionally, the two tasks used in this work have an incremental relationship: standing is easier than walking. Testing our approach on tasks of equal difficulty could provide further insights into its performance.

Another limitation involves the reward function. The walking agent consistently outperformed the standing agent across both tasks. This was due to the reward function in the standing task which does not penalize movement. This meant that there was no negative transfer from walking to standing. To better reflect a realistic multi-tasking setting, future work should consider adding a negative reward for movement in the standing task.

Lastly, we collected data using a fixed sample size, which does not test how robust our approach is to varying amounts of data. Investigating the impact of different dataset sizes on performance would be an interesting direction for future work. Understanding how the approach performs with less data could provide useful insights into its applicability in real-world problems.

8

Conclusion

This research focused on Policy Distillation (PD) in the Offline Multi-Task Reinforcement Learning (MTRL) setting. Specifically, we addressed the following two research questions:

1. How can PD operate effectively in the Offline MTRL setting?
2. How can the common structure across related tasks be better captured through PD?

To answer these questions, we introduced a general approach called Multi-Task Offline Policy Distillation (MOP). MOP involves learning teacher policies from offline data using an offline RL algorithm and distilling this knowledge into a student policy. The distillation process involves sampling states from the offline data, feeding them to the teacher and student networks, and comparing the teachers' outputs to the student's outputs using a distillation loss function. The student is then optimized by minimizing this loss. Specific instances and adaptations of MOP were explored to address each of the research questions. In this chapter, we summarize our work and present the main conclusions.

8.1. Operating Policy Distillation Offline

To address the first research question, we established a standard setup of MOP. We utilized the Pessimistic Bootstrapping for Offline RL (PBRL) algorithm to learn the teacher policies. The teachers trained with PBRL closely matched or slightly exceeded the performance of their corresponding behavior policies. We also demonstrated that direct multi-tasking using PBRL is prone to negative transfer, justifying the need for PD.

We explored three different distillation loss functions: Mean Squared Error (MSE) loss, Q-value loss, and KL-divergence. The results showed that direct optimization with MSE loss was the most effective. The student was able to match the performance of its teachers across tasks, while being able to perform all tasks using the same network size as a single teacher. This demonstrates that MOP can effectively merge multiple task-specific policies into a single multi-task policy in the offline RL setting.

Next, we examined the impact of a state distribution shift between the offline data and the teacher policy on the performance of our approach. Addressing distribution shift is a major challenge in offline RL, so it was essential to assess MOP's robustness in this regard. We hypothesized that a larger distribution shift would require more capacity in the student network (Hypothesis 1). We performed a series of experiments to test this hypothesis. Our findings suggest that, in general, a larger distribution shift degrades the student's performance when the network size remains fixed. However, we observed that a small distribution shift can actually be beneficial for small networks, as it introduces more diversity into the offline data, potentially enhancing out-of-distribution robustness. This kind of robustness is particularly beneficial for very small networks, which are more prone to selecting suboptimal actions that lead the agent into out-of-distribution states.

Moreover, increasing the network size can mitigate the effects of a large distribution shift and improve performance, but it is not always sufficient to close the gap between the student and teacher performance

when the shift is too large. This is because simply increasing network capacity is prone to overfitting to the offline data. Therefore, we recommend that future work focuses on developing more sophisticated methods for mitigating the distribution shift to enhance student performance in these situations.

8.2. Capturing Common Structure Through Policy Distillation

Addressing the first research question led to an instance of MOP that could effectively merge single-task teacher policies into a multi-task student policy, achieving comparable performance while maintaining the size of a single teacher network. In the second research question, we investigated the fundamental concept of MTRL: sharing common structure across tasks while separating task-specific properties. To explore this, we formally defined common structure at two levels: the trajectory level and the computational level. Common structure at the trajectory level refers to cases where the optimal action for a given state is the same across tasks. In contrast, common structure at the computational level refers to shared computation in the student network across tasks, independent of specific state-action pairs.

To get an indication of the amount of common structure captured, we designed a method to measure shared computation across tasks. This metric evaluates the correlation between neuron activation values and the task ID. We proposed two approaches for comparing activation values across tasks: per-state and average comparisons. The per-state approach compares activations for identical input states under both task IDs, while the average approach compares the mean activation values over each task's state distribution. Both approaches have their limitations. A per-state comparison does not account for differences in state distributions between tasks, while the average comparison has the risk that activation values cancel out against each other. We quantified similarity using Kendall's τ, which ranks neuron activity for each task and compares these rankings across tasks. This intuitively indicates whether the same neurons are consistently active across tasks, providing an estimate of the common structure captured. However, comparing neurons by relative activity alone may not fully capture the concept of common structure. Future work should establish invariance properties for a robust metric of common structure. Additionally, techniques for comparing activation values that account for each task's state distribution should be further explored.

The results of measuring shared computation showed that MOP initially shared a considerable amount of computation across tasks, particularly when the teacher policies were of expert level. For medium-level teacher policies, the degree of sharing was lower. We hypothesize that expert policies display more aligned behavior in common structure states due to their optimality, enabling PD to merge these behaviors more effectively.

At the computational level, we made two attempts to increase shared structure across tasks. First, we reduced the network size. This was intended to create an information bottleneck, forcing the student network to merge redundant behaviors to maintain performance. However, the absence of an explicit incentive led to degraded performance and less shared computation. Next, we introduced ranking regularization. This approach added a regularization term to the loss function to explicitly encourage greater shared computation across tasks. We experimented with three types of regularization losses: pairwise logistic loss, listwise softmax cross-entropy loss, and Normalized Discounted Cumulative Gain (NDCG) loss. The pairwise loss increased sharing but caused the network to ignore the task ID, resulting in negative transfer and reduced performance. The listwise and NDCG losses maintained performance but did not increase shared computation. We concluded that the abstract nature of common structure at the computational level makes it difficult to define a clear objective, let alone to optimize it. This highlights the need for a standardized, formal definition of common structure at the computational level.

To address common structure at the trajectory level, we hypothesized that PD can only share behaviors across tasks when the teachers exhibit identical behaviors in common structure states. However, such alignment is not always present. Teacher behaviors may be unaligned if at least one teacher is suboptimal or if multiple equally optimal but distinct behaviors exist. We argued that multi-task exploration is necessary to align teacher behaviors. This would require evaluating behaviors from one task in the context of another task. However, in the offline RL setting, no interaction with the environment is possible during training. The agent is limited to learning from previously collected data, which restricts exploration.

To address this, we introduced the assumption that the reward functions are known to the agent, allowing for multi-task exploration. Under this assumption, we proposed two extensions to MOP: Data Sharing (DS) and Offline Q-Switch (OQS).

DS involves relabeling and sharing all data across tasks, resulting in a mixed offline dataset for each task, which is then used to train the teacher policies. This theoretically improves performance by expanding the coverage of the offline data. Additionally, we expected teacher behaviors to naturally align across tasks. This is because the teachers were trained on the same data, and our environment consists of a continuous state-action space, making it highly unlikely that distinct, equally optimal behaviors would be learned. Results showed that DS could improve teacher performance in specific situations, but performance significantly degraded when the state distribution shift increased. The student's performance matched that of the teachers, and therefore the student was also indirectly affected by this exacerbated distribution shift. However, the amount of shared computation increased significantly, indicating that multi-task exploration does help align teacher behaviors.

OQS also involves relabeling data but keeps data from other tasks in separate datasets. These separate datasets are used to train cross-task teachers. During distillation, each cross-task teacher computes its best action along with its corresponding Q-value, and the student mimics the action with the highest Q-value. We expected the student to perform at least as well as the best cross-task teacher or the original teacher. While shared computation increased, the overall student performance declined. This was likely due to insufficient coordination between the teachers' Q-values, resulting in suboptimal decisions across entire trajectories.

In conclusion, MOP initially captured some common structure, as demonstrated by the high correlation between task ID and neuron activation values. We attempted to improve the sharing of common structure at the computational level by reducing the network size and ranking regularization. Nonetheless, this failed due to the abstract nature of common structure at the computational level. It may not even exist in a clearly defined form. We then aimed to capture common structure at the trajectory level by introducing multi-task exploration. However, this is particularly challenging in the offline setting, where exploration is not possible. The two extensions we introduced, DS and OQS, allowed for some form of multi-task exploration under the assumption of known reward functions. While these methods did successfully increase shared computation, they did not consistently improve performance. This highlights the need for further research into effective multi-task exploration techniques in the offline RL setting.

8.3. Future Work

Throughout this work, we have proposed various directions for future research. In this section, we provide an overview of the most important areas for further investigation, along with high-level ideas for potential approaches.

First, a valuable direction for future research would be to explore methods to mitigate the state distribution shift during state sampling in the distillation process. One promising idea is to apply reweighting techniques, similar to Importance Sampling (IS) [58], by using the ratio of Q-values from the behavior and teacher policies. This should serve as an approximation for the probability density ratio. This approach could help prioritize states which the teacher policy is more likely to encounter in the environment. However, it relies on access to the Q-value function of the behavior policy. This is typically unavailable in offline RL, but could be estimated through Behavioral Cloning (BC) [35] for example.

Furthermore, establishing a standardized, formal definition of common structure is essential. Related work does not formally define common structure, making it challenging to compare MTRL methods. Defining common structure at the computational level is particularly difficult due to its abstract nature. We can intuitively relate it to human skills, such as a notion of 'ball feeling' when playing different sports like football and tennis. However, mathematically expressing this concept requires a thorough examination of its properties and even consideration of whether it truly exists. Without a standardized definition, it is challenging not only to compare methods but also to define a concrete objective for optimization.

Moreover, our metric for measuring shared computation as an indicator of the amount of captured

common structure should be refined. Future work should investigate methods to compare activation values while accounting for task-specific state distributions. Additionally, appropriate invariances that align with a formal definition of common structure should be explored and incorporated into the similarity measure. Establishing such invariances would help ensure that the measured values accurately correspond to the true amount of common structure shared across tasks.

Additionally, exploring methods to generate imaginary trajectories for multi-task exploration presents a promising direction. Techniques like Skills Regularized Task Decomposition (SRTD) [74] generate trajectories using a latent space, leveraging shared skills across tasks. However, alignment of the skills across tasks may still be limited by the accuracy of the learned latent space. Incorporating context from Large Language Models (LLMs) [1] could enhance this approach by providing broader contextual understanding. For example, methods such as Knowledgeable Agents from Language Model Rollouts (KALM) [47] use textual descriptions to create diverse, aligned trajectories across tasks. Applying LLMs to generate more accurate task-aligned behaviors could significantly improve multi-task exploration in offline RL.

Lastly, our experiments highlighted limitations within the DeepMind-control Walker [59] environment. Deploying the walking teacher in the stand task consistently outperformed the standing teacher, suggesting it would be more effective to simply copy the walking teacher for both tasks rather than learning a true multi-task student. Future work could address this by introducing a movement penalty in the stand task to encourage distinct task-specific behaviors.

Acknowledgments

I would like to thank MSc. D. Mambelli for his continuous support and for our engaging discussions that enriched this project. I am also grateful to Prof. dr. M.T.J. Spaan for his constructive feedback and valuable advice on high-level decisions throughout the project. Furthermore, I would like to thank the Delft High Performance Computing Centre (DHPC) for providing access to their computing cluster, which offered excellent and reliable service for my computational needs.

Bibliography

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. "GPT-4 Technical Report". In: *arXiv preprint arXiv:2303.08774* (2023).

[2] M. Arjovsky, L. Bottou, I. Gulrajani, and D. Lopez-Paz. "Invariant Risk Minimization". In: *arXiv preprint arXiv:1907.02893* (2019).

[3] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. "Deep Reinforcement Learning: A Brief Survey". In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38.

[4] C. Bai, L. Wang, J. Hao, Z. Yang, B. Zhao, Z. Wang, and X. Li. "Pessimistic Value Iteration for Multi-Task Data Sharing in Offline Reinforcement Learning". In: *Artificial Intelligence* 326 (2024), p. 104048. ISSN: 0004-3702.

[5] C. Bai, L. Wang, Z. Yang, Z. Deng, A. Garg, P. Liu, and Z. Wang. "Pessimistic Bootstrapping for Uncertainty-Driven Offline Reinforcement Learning". In: *arXiv preprint arXiv:2202.11566* (2022).

[6] J. Beck, R. Vuorio, E. Z. Liu, Z. Xiong, L. Zintgraf, C. Finn, and S. Whiteson. "A Survey of Meta-Reinforcement Learning". In: *arXiv preprint arXiv:2301.08028* (2023).

[7] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, et al. "Dota 2 with Large Scale Deep Reinforcement Learning". In: *arXiv preprint arXiv:1912.06680* (2019).

[8] D. Borsa, T. Graepel, and J. Shawe-Taylor. "Learning Shared Representations in Multi-Task Reinforcement Learning". In: *arXiv preprint arXiv:1603.02041* (2016).

[9] S. Bruch, X. Wang, M. Bendersky, and M. Najork. "An Analysis of the Softmax Cross Entropy Loss for Learning-to-Rank with Binary Relevance". In: *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval*. 2019, pp. 75–78.

[10] E. Brunskill and L. Li. "Sample Complexity of Multi-Task Reinforcement Learning". In: *arXiv preprint arXiv:1309.6821* (2013).

[11] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. "Learning to Rank Using Gradient Descent". In: *Proceedings of the 22nd International Conference on Machine Learning*. 2005, pp. 89–96.

[12] Y. Chebotar, Q. Vuong, K. Hausman, F. Xia, Y. Lu, A. Irpan, A. Kumar, T. Yu, A. Herzog, K. Pertsch, et al. "Q-Transformer: Scalable Offline Reinforcement Learning via Autoregressive Q-Functions". In: *Conference on Robot Learning (CoRL)*. PMLR. 2023, pp. 3909–3928.

[13] S. Chowdhuri, T. Pankaj, and K. Zipser. "MultiNet: Multi-Modal Multi-Task Learning for Autonomous Driving". In: *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2019, pp. 1496–1504.

[14] C. D'Eramo, D. Tateo, A. Bonarini, M. Restelli, and J. Peters. "Sharing Knowledge in Multi-Task Deep Reinforcement Learning". In: *arXiv preprint arXiv:2401.09561* (2024).

[15] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. "Benchmarking Deep Reinforcement Learning for Continuous Control". In: *International Conference on Machine Learning (ICML)*. PMLR. 2016, pp. 1329–1338.

[16] J. Duchi. *Derivations for Linear Algebra and Optimization*. University of California, Berkeley. 2014.

[17] R. Figueiredo Prudencio, M. R. O. A. Maximo, and E. L. Colombini. "A Survey on Offline Reinforcement Learning: Taxonomy, Review, and Open Problems". In: *IEEE Transactions on Neural Networks and Learning Systems* 35.8 (2024), pp. 10237–10257.

[18] D. Freedman and P. Diaconis. "On the Histogram as a Density Estimator: L2 Theory". In: *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete* 57.4 (1981), pp. 453–476.

[19] S. Fujimoto, H. Hoof, and D. Meger. "Addressing Function Approximation Error in Actor-Critic Methods". In: *International Conference on Machine Learning (ICML)*. PMLR. 2018, pp. 1587–1596.

[20] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska. "A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.6 (2012), pp. 1291–1307.

[21] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. "Continuous Deep Q-Learning with Model-Based Acceleration". In: *International Conference on Machine Learning (ICML)*. PMLR. 2016, pp. 2829–2838.

[22] A. Hallak, D. Di Castro, and S. Mannor. "Contextual Markov Decision Processes". In: *arXiv preprint arXiv:1502.02259* (2015).

[23] K. Hara, D. Saito, and H. Shouno. "Analysis of Function of Rectified Linear Unit Used in Deep Learning". In: *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2015, pp. 1–8.

[24] T. Hastie, R. Tibshirani, and J. Friedman. "Overview of Supervised Learning". In: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (2009), pp. 9–41.

[25] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. "Deep Reinforcement Learning That Matters". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[26] M. Hessel, H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt, and H. Van Hasselt. "Multi-Task Deep Reinforcement Learning with PopArt". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 3796–3803.

[27] G. Hinton, O. Vinyals, and J. Dean. *Distilling the Knowledge in a Neural Network*. 2015.

[28] J. J. Hopfield. "Neural Networks and Physical Systems with Emergent Collective Computational Abilities". In: *Proceedings of the National Academy of Sciences* 79.8 (1982), pp. 2554–2558.

[29] S. Jayanthi, L. Chen, N. Balabanska, V. Duong, E. Scarlatescu, E. Ameperosa, Z. H. Zaidi, D. Martin, T. K. Del Matto, M. Ono, et al. "DROID: Learning from Offline Heterogeneous Demonstrations via Reward-Policy Distillation". In: *Conference on Robot Learning (CoRL)*. PMLR. 2023, pp. 1547–1571.

[30] L. P. Kaelbling, M. L. Littman, and A. W. Moore. "Reinforcement Learning: A Survey". In: *Journal of Artificial Intelligence Research* 4 (1996), pp. 237–285.

[31] 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 (CoRL)*. PMLR. 2018, pp. 651–673.

[32] D. Kalashnikov, J. Varley, Y. Chebotar, B. Swanson, R. Jonschkowski, C. Finn, S. Levine, and K. Hausman. "MT-OPT: Continuous Multi-Task Robotic Reinforcement Learning at Scale". In: *arXiv preprint arXiv:2104.08212* (2021).

[33] S. Kılıçarslan, K. Adem, and M. Çelik. "An Overview of the Activation Functions Used in Deep Learning Algorithms". In: *Journal of New Results in Science* 10.3 (2021), pp. 75–88.

[34] S. Kornblith, M. Norouzi, H. Lee, and G. Hinton. "Similarity of Neural Network Representations Revisited". In: *International Conference on Machine Learning (ICML)*. PMLR. 2019, pp. 3519–3529.

[35] A. Kumar, J. Hong, A. Singh, and S. Levine. "Should I Run Offline Reinforcement Learning or Behavioral Cloning?" In: *International Conference on Learning Representations (ICLR)*. 2021.

[36] A. Kumar, A. Singh, F. Ebert, M. Nakamoto, Y. Yang, C. Finn, and S. Levine. "Pre-Training for Robots: Offline RL Enables Learning New Tasks from a Handful of Trials". In: *arXiv preprint arXiv:2210.05178* (2022).

[37] A. Kumar, A. Zhou, G. Tucker, and S. Levine. "Conservative Q-Learning for Offline Reinforcement Learning". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 1179–1191.

[38] S. Levine, A. Kumar, G. Tucker, and J. Fu. "Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems". In: *CoRR* abs/2005.01643 (2020).

[39] T. Lillicrap. "Continuous Control with Deep Reinforcement Learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[40] Y. Liu, S. Liang, H. Wang, Y. Liang, and A. Gitter. *Avoiding Negative Transfer on a Focused Task with Deep Multi-Task Reinforcement Learning*. 2017.

[41] K. Lu, S. Zhang, and X. Chen. "Goal-Oriented Dialogue Policy Learning from Failures". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 2596–2603.

[42] B. Mirchevska, M. Werling, and J. Boedecker. "Optimizing Trajectories for Highway Driving with Offline Reinforcement Learning". In: *Frontiers in Future Transportation* 4 (2023), p. 1076439.

[43] V. Mnih. "Playing Atari with Deep Reinforcement Learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[44] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. "Human-Level Control Through Deep Reinforcement Learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[45] A. Mohan, A. Zhang, and M. Lindauer. "Structure in Deep Reinforcement Learning: A Survey and Open Problems". In: *Journal of Artificial Intelligence Research* 79 (2024), pp. 1167–1236.

[46] K. Ota, D. K. Jha, and A. Kanezaki. "A Framework for Training Larger Networks for Deep Reinforcement Learning". In: *Machine Learning* (2024), pp. 1–25.

[47] J.-C. Pang, S.-H. Yang, K. Li, J. Zhang, X.-H. Chen, N. Tang, and Y. Yu. "Knowledgeable Agents by Offline Reinforcement Learning from Large Language Model Rollouts". In: *arXiv preprint arXiv:2404.09248* (2024).

[48] E. Parisotto, J. L. Ba, and R. Salakhutdinov. "Actor-Mimic: Deep Multitask and Transfer Reinforcement Learning". In: *arXiv preprint arXiv:1511.06342* (2015).

[49] M. L. Puterman. "Markov Decision Processes". In: *Handbooks in Operations Research and Management Science* 2 (1990), pp. 331–434.

[50] E. Rosete-Beas, O. Mees, G. Kalweit, J. Boedecker, and W. Burgard. "Latent Plans for Task-Agnostic Offline Reinforcement Learning". In: *Conference on Robot Learning (CoRL)*. PMLR. 2023, pp. 1838–1849.

[51] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell. "Policy Distillation". In: *arXiv preprint arXiv:1511.06295* (2015).

[52] O. Scheel, L. Bergamini, M. Wolczyk, B. Osiński, and P. Ondruska. "Urban Driver: Learning to Drive from Real-World Demonstrations Using Policy Gradients". In: *Conference on Robot Learning (CoRL)*. PMLR. 2022, pp. 718–728.

[53] S. Siegel and N. J. Castellan. *Nonparametric Statistics for the Behavioral Sciences*. 2nd. New York: McGraw-Hill, 1988, pp. 213–214.

[54] S. Sinha, H. Bharadhwaj, A. Srinivas, and A. Garg. "D2RL: Deep Dense Architectures in Reinforcement Learning". In: *arXiv preprint arXiv:2010.09163* (2020).

[55] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018.

[56] J. Swentworth. *Fixing the Good Regulator Theorem*. Feb. 2021. URL: `https://www.lesswrong.com/posts/Dx9LoqsEh3gHNJMDk/fixing-the-good-regulator-theorem`.

[57] C. Szepesvári and M. L. Littman. "A Unified Analysis of Value-Function-Based Reinforcement-Learning Algorithms". In: *Neural Computation* 11.8 (1999), pp. 2017–2060.

[58] A. Tabandeh, G. Jia, and P. Gardoni. "A Review and Assessment of Importance Sampling Methods for Reliability Analysis". In: *Structural Safety* 97 (2022), p. 102216.

[59] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. de Las Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, T. P. Lillicrap, and M. A. Riedmiller. "DeepMind Control Suite". In: *CoRR* abs/1801.00690 (2018).

[60] M. E. Taylor and P. Stone. "Transfer Learning for Reinforcement Learning Domains: A Survey". In: *Journal of Machine Learning Research* 10.7 (2009).

[61] Y. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu. "Distral: Robust Multitask Reinforcement Learning". In: *Advances in Neural Information Processing Systems* 30 (2017).

[62] E. Todorov, T. Erez, and Y. Tassa. "MuJoCo: A Physics Engine for Model-Based Control". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2012, pp. 5026–5033. ISBN: 978-1-4673-1737-5.

[63] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, et al. "StarCraft II: A New Challenge for Reinforcement Learning". In: *arXiv preprint arXiv:1708.04782* (2017).

[64] N. Vithayathil Varghese and Q. H. Mahmoud. "A Survey of Multi-Task Deep Reinforcement Learning". In: *Electronics* 9.9 (2020), p. 1363.

[65] Y. Wang, L. Wang, Y. Li, D. He, W. Chen, and T. Liu. "A Theoretical Analysis of Normalized Discounted Cumulative Gain (NDCG) Ranking Measures". In: *Proceedings of the 26th Annual Conference on Learning Theory (COLT)*. Citeseer. 2013.

[66] C. J. Watkins and P. Dayan. "Q-Learning". In: *Machine Learning* 8 (1992), pp. 279–292.

[67] S. Węglarczyk. "Kernel Density Estimation and Its Application". In: *ITM Web of Conferences*. Vol. 23. EDP Sciences. 2018, p. 00037.

[68] R. J. Williams. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Machine Learning* 8 (1992), pp. 229–256.

[69] Y. Wu, G. Tucker, and O. Nachum. "Behavior Regularized Offline Reinforcement Learning". In: *arXiv preprint arXiv:1911.11361* (2019).

[70] Z. Xu, K. Wu, Z. Che, J. Tang, and J. Ye. "Knowledge Transfer in Multi-Task Deep Reinforcement Learning for Continuous Control". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 15146–15155.

[71] S. Yao, R. Rao, M. Hausknecht, and K. Narasimhan. "Keep Calm and Explore: Language Models for Action Generation in Text-Based Games". In: *arXiv preprint arXiv:2010.02903* (2020).

[72] E. Yilmaz, J. A. Aslam, and S. Robertson. "A New Rank Correlation Coefficient for Information Retrieval". In: *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2008, pp. 587–594.

[73] X. Ying. "An Overview of Overfitting and Its Solutions". In: *Journal of Physics: Conference Series*. Vol. 1168. IOP Publishing. 2019, p. 022022.

[74] M. Yoo, S. Cho, and H. Woo. "Skills Regularized Task Decomposition for Multi-Task Offline Reinforcement Learning". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 37432–37444.

[75] T. Yu, A. Kumar, Y. Chebotar, K. Hausman, C. Finn, and S. Levine. "How to Leverage Unlabeled Data in Offline Reinforcement Learning". In: *International Conference on Machine Learning (ICML)*. PMLR. 2022, pp. 25611–25635.

[76] T. Yu, A. Kumar, Y. Chebotar, K. Hausman, S. Levine, and C. Finn. "Conservative Data Sharing for Multi-Task Offline Reinforcement Learning". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 11501–11516.

[77] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn. "Gradient Surgery for Multi-Task Learning". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 5824–5836.

[78] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. "Understanding Deep Learning (Still) Requires Rethinking Generalization". In: *Communications of the ACM* 64.3 (2021), pp. 107–115.

[79] C. Zhang, O. Vinyals, R. Munos, and S. Bengio. "A Study on Overfitting in Deep Reinforcement Learning". In: *arXiv preprint arXiv:1804.06893* (2018).

[80] G. Zhang, A. Jain, I. Hwang, S.-H. Sun, and J. J. Lim. "Efficient Multi-Task Reinforcement Learning via Selective Behavior Sharing". In: *arXiv preprint arXiv:2302.00671* (2023).

A

TD3 Agent Specifications

- Learning rate: 10^{-4}
- Standard deviation in clipped noise: $3 \cdot 10^{-1}$
- Actor/critic update ratio: $\frac{1}{2}$
- Critic target smoothing rate: 10^{-2}
- Neural network properties:
  - Fully-connected feed forward
  - Input layer of 24 neurons (corresponding to the state dimensions of the Walker environment)
  - Two hidden layers of 1024 neurons each
  - Output layer of 6 neurons (corresponding to the action dimensions of the Walker environment)

B

PBRL Agent Specifications

- Learning rate: 10^{-4}
- Standard deviation in clipped noise: $2 \cdot 10^{-1}$
- Actor/critic update ratio: $\frac{1}{2}$
- Critic target smoothing rate: $5 \cdot 10^{-3}$
- Actor target smoothing rate: $5 \cdot 10^{-3}$
- Neural network properties:
  - Fully-connected feed forward
  - Input layer of 24 neurons (corresponding to the state dimensions of the Walker environment)
  - Two hidden layers of 256 neurons each
  - Output layer of 6 neurons (corresponding to the action dimensions of the Walker environment)

C

Single-Task PBRL Results

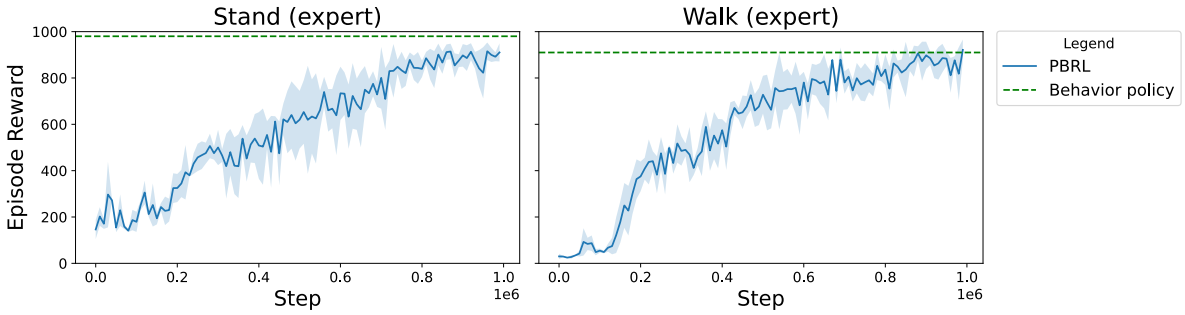


Figure C.1: Results of PBRL on the expert offline datasets.

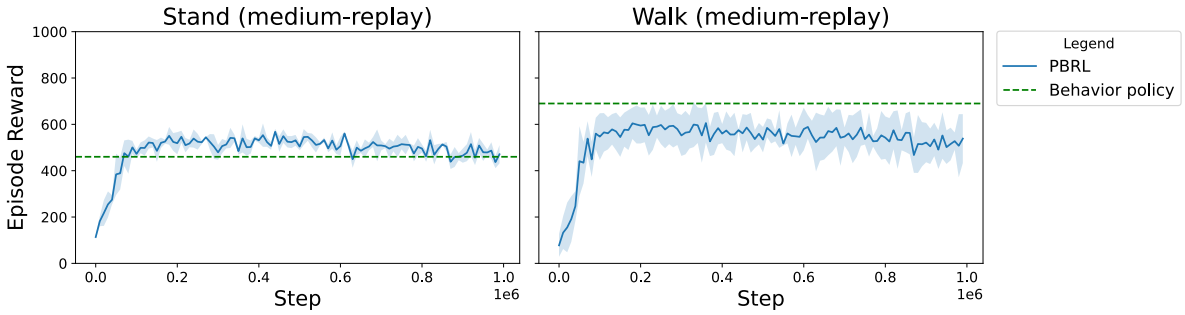


Figure C.2: Results of PBRL on the medium-replay offline datasets.

D

Naive Multi-Task PBRL Results

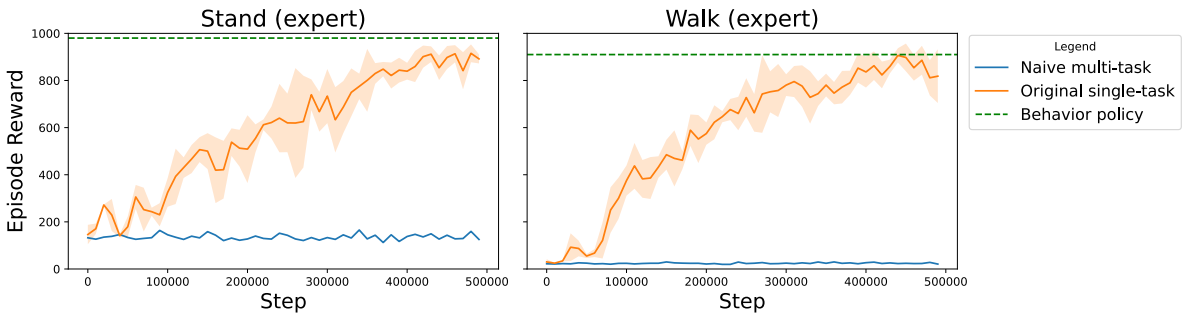


Figure D.1: Results of naive multi-task and original single-task PBRL on the expert offline datasets.

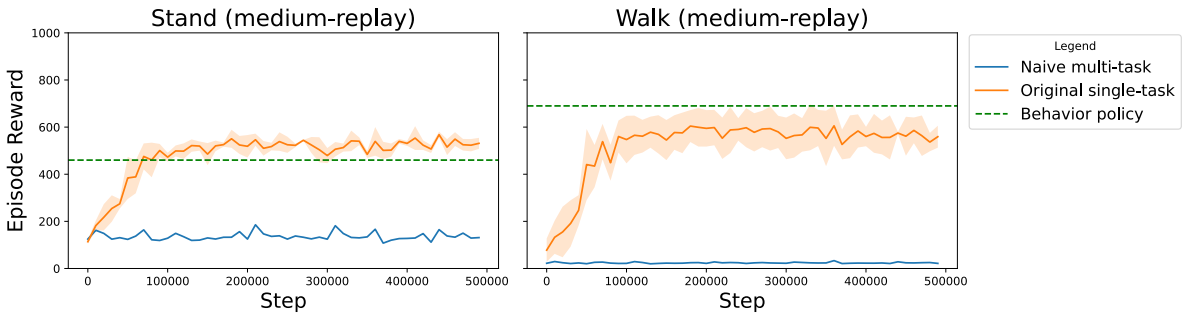


Figure D.2: Results of naive multi-task and original single-task PBRL on the medium-replay offline datasets.

E

MSE Loss Results

Below the results are plotted for all combination of datasets for MOP with MSE loss $\mathcal{L}_{MSE}$.

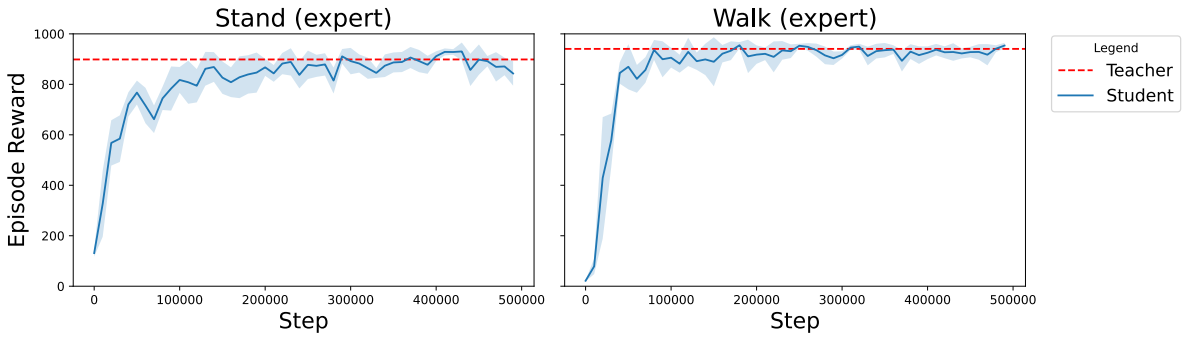


Figure E.1: Expert X Expert

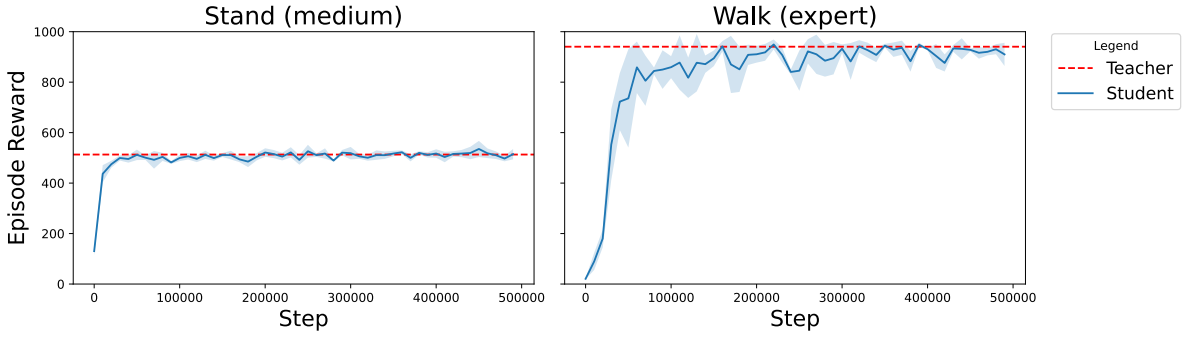


Figure E.2: Expert X Medium

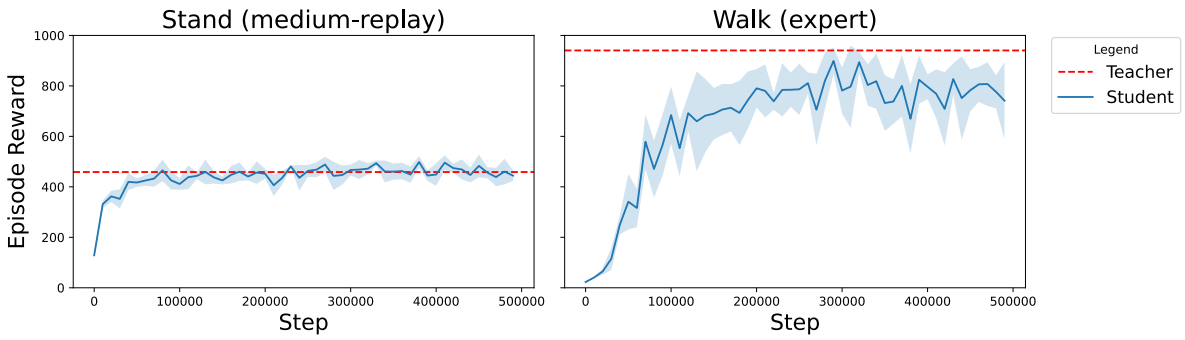


Figure E.3: Expert X Medium-replay

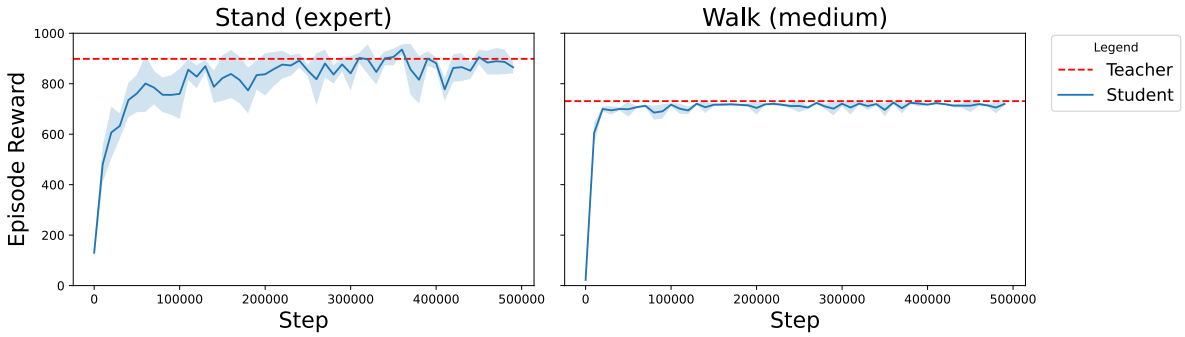


Figure E.4: Medium X Expert

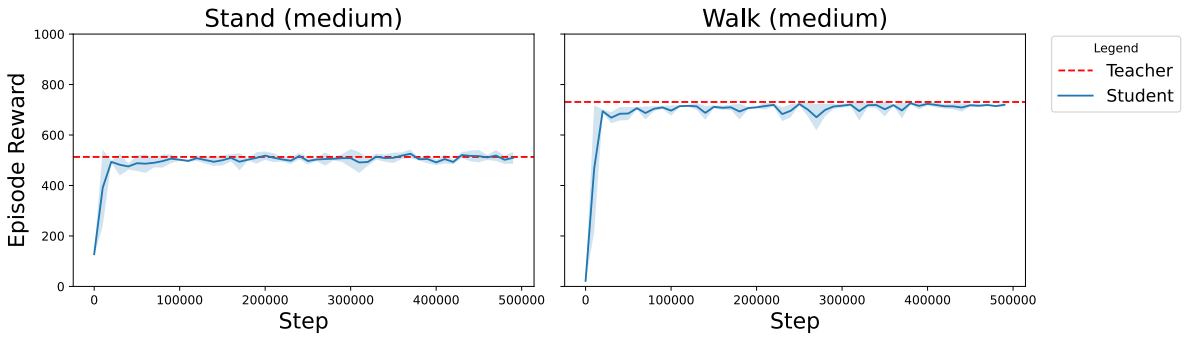


Figure E.5: Medium X Medium

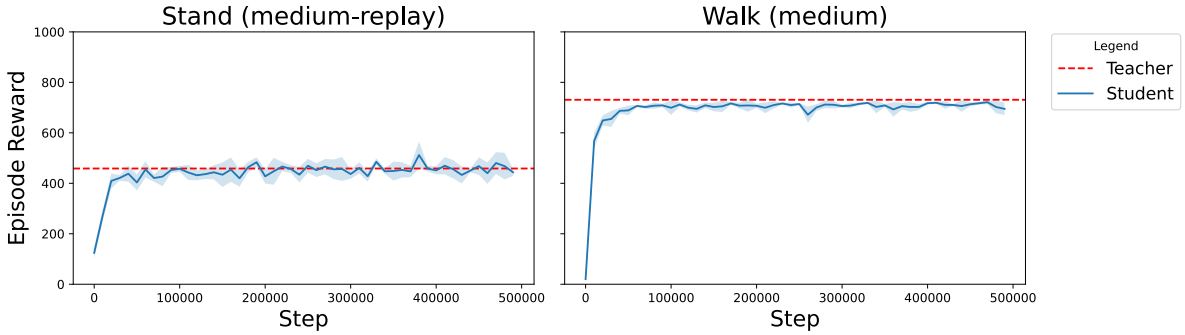


Figure E.6: Medium X Medium-replay

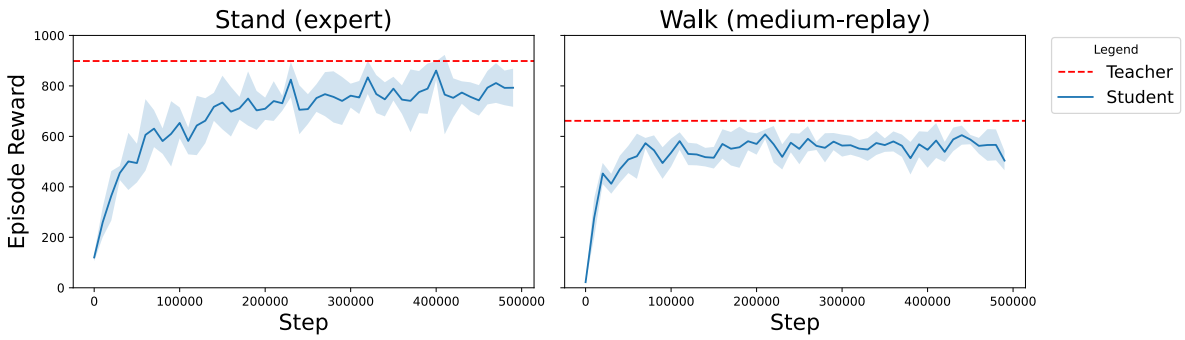


Figure E.7: Medium-replay X Expert

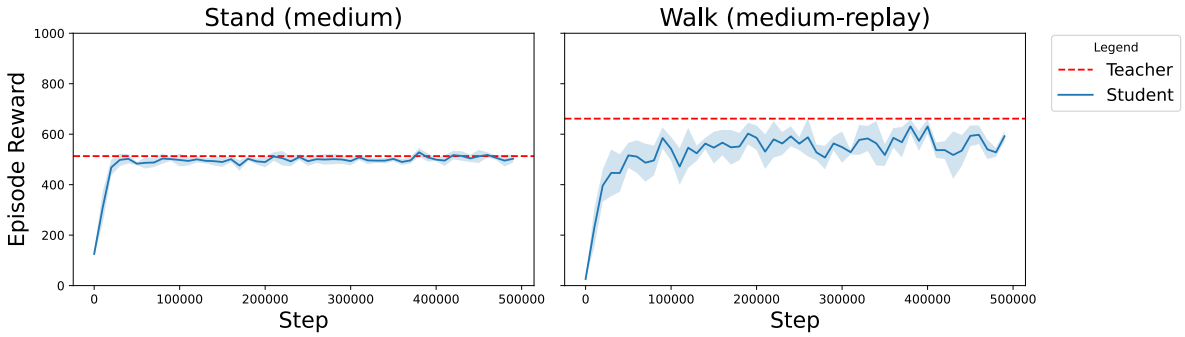


Figure E.8: Medium-replay X Medium

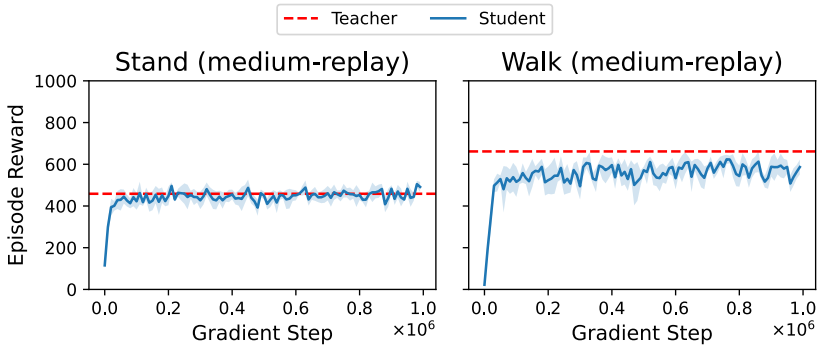
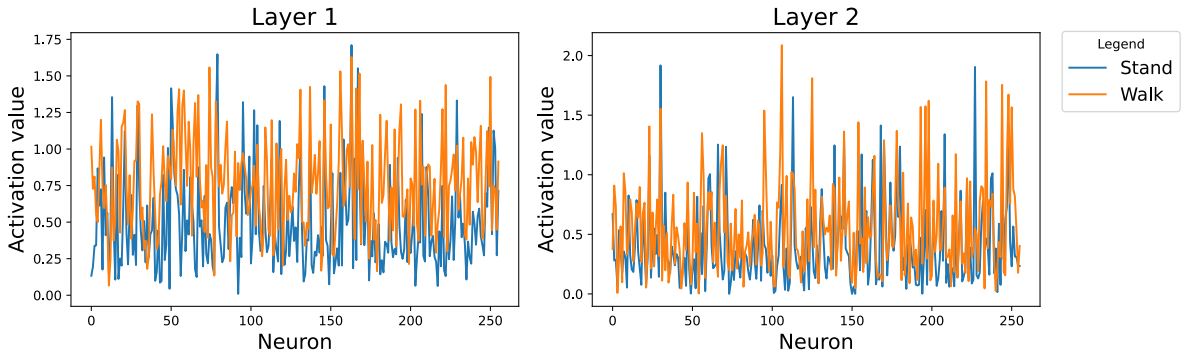


Figure E.9: Medium-replay X Medium-replay

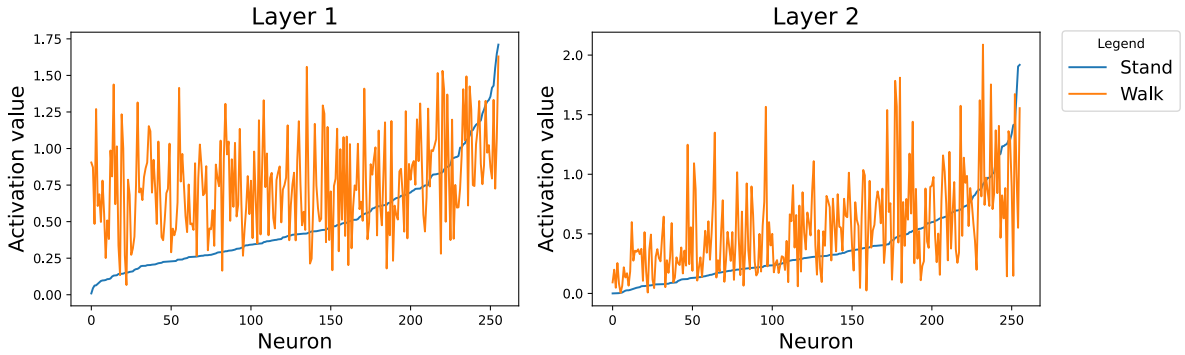
F

Measuring Shared Computation Results

F.1. Expert Offline Datasets



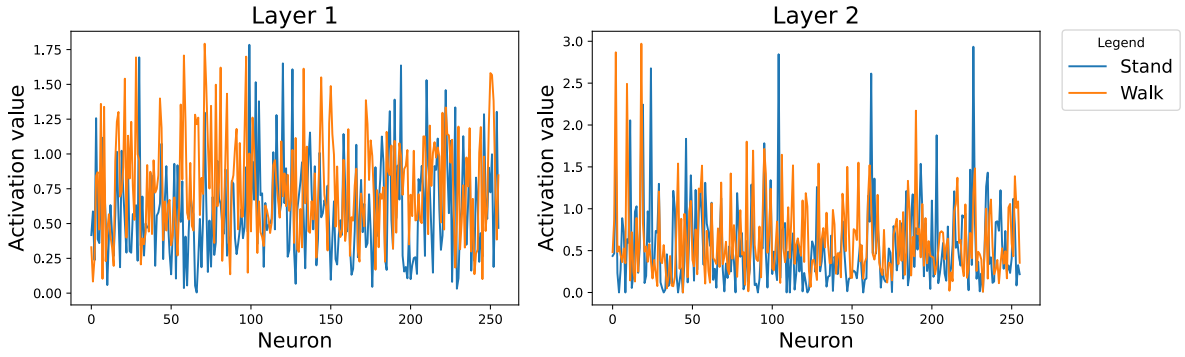
(a) Unsorted.



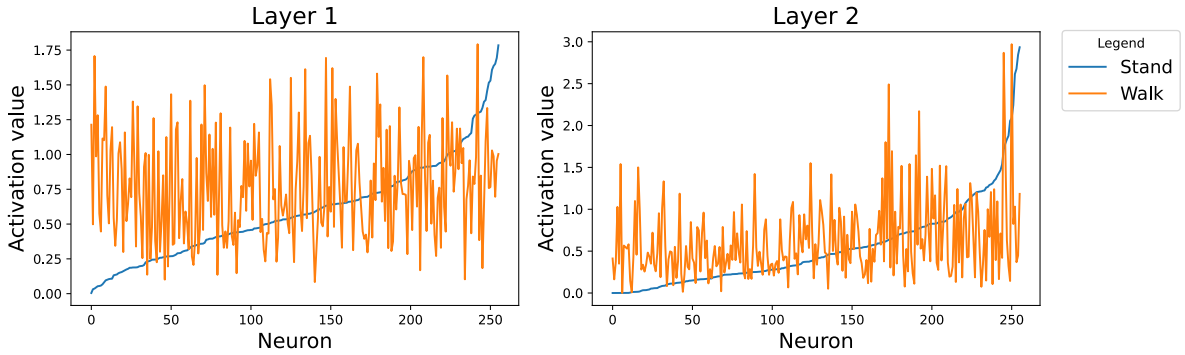
(b) Sorted by the average activation values of the standing task in ascending order.

Figure F.1: The average activation values per neuron of the student network trained on the Expert datasets after being deployed online to perform the standing and walking tasks.

F.2. Medium Offline Datasets



(a) Unsorted.



(b) Sorted by the average activation values of the standing task in ascending order.

Figure F.2: The average activation values per neuron of the student network trained on the Medium datasets after being deployed online to perform the standing and walking tasks.