university of groningen / faculty of science and engineering / artificial intelligence

Transfer Learning in Reinforcement Learning: When Task-Specific Adaptation Outperforms Generalization

Bachelor's Project Thesis

Cătălin Zaharia, S4625765, c.z.catalin.zaharia@student.rug.nl,
Supervisor: Rafael Fernandes Cunha, r.f.cunha@rug.nl

Abstract: Transfer Learning has emerged as a transformative technique in Reinforcement Learning (RL), enabling algorithms to leverage knowledge from prior tasks to improve efficiency and generalization. This study investigates whether transfer learning can further enhance the already exceptional learning capabilities of the Parallelized Q-Network (PQN). Using the MinAtar environment as a controlled testbed, I integrate transfer learning techniques to assess their impact on learning time. By comparing the algorithm with and without transfer learning across multiple tasks, the results reveal that transfer learning consistently underperforms when compared to task-specific training. Contrary to expectations, transfer configurations fail to accelerate learning, with environment-specific training emerging as the superior approach. These findings underscore the limitations of transfer learning in RL and reaffirm the critical role of environment-specific adaptation in achieving efficient and robust learning outcomes.

1 Introduction

The growing capabilities of GPU-based computing have revolutionized **Reinforcement Learning** (RL), enabling researchers to scale algorithms and explore complex environments with unprecedented efficiency. GPU-accelerated libraries and frameworks such as `JAX` [14], alongside benchmarks [3, 5, 8, 9], offer high-performance simulations tailored for RL. These tools allow seamless integration of large-scale parallelism, making it easier to execute computationally intensive tasks across multiple environments. These benchmarks and frameworks not only accelerate training but also facilitate novel algorithmic advancements by supporting diverse and resource-demanding tasks. This technological shift underscores the need for RL architectures capable of fully exploiting GPU potential.

One such architecture is the **Parallelized Q-Network** (PQN) [2], which exemplifies the application of GPU-optimized RL. PQN simplifies deep temporal difference learning by removing replay buffers and target networks, achieving stability and speed through synchronous updates across parallel environments. Leveraging the power of GPUs, PQN efficiently utilizes vectorized environments and incorporates regularization techniques such as LayerNorm [6] or BatchNorm [4] to ensure reliable performance. These design choices make PQN a candidate for scalable RL, excelling in tasks requiring rapid learning and robust convergence.

While PQN demonstrates exceptional efficiency, the question arises whether **Transfer Learning** (TL) can further enhance its already fast learning capabilities. TL, a cornerstone of machine learning, enables algorithms to reuse knowledge from prior tasks to accelerate learning in new scenarios. However, the applicability of TL in RL, particularly in model-free settings, remains challenging. Surveys such as [15] and [22] highlight key difficulties, including negative transfer caused by task dissimilarities, mismatched state-action spaces, and insufficient compatibility between source and target environments. These challenges underscore the importance of carefully designing transfer mechanisms to balance adaptability and robustness. Despite these obstacles, TL has demonstrated potential in RL, particularly through methods such as fine-tuning pre-trained networks, leveraging modu-

lar components, or utilizing shared representations. Advanced techniques like fractional TL [16] and universal feature spaces [13] optimize knowledge transfer across tasks while addressing task-specific variability.

In this study, we investigate the impact of TL on the learning time of the Parallelized Q-Network in the *MinAtar* [21] environment, a controlled and lightweight testbed designed for RL research. By integrating TL techniques into PQN, we aim to evaluate whether knowledge reuse can complement PQN's inherent speed and stability. Through a series of experiments comparing PQN with and without TL across multiple tasks, we assess the potential for TL to further optimize learning time in advanced RL frameworks.

The results of this research provide insights into the applicability of TL for scalable RL architectures, contributing to a deeper understanding of how modular knowledge transfer techniques can benefit GPU-optimized systems. In this work, I summarize my empirical contributions as follows: I) I demonstrate that TL configurations frequently fail to outperform or even match the baseline, highlighting the limitations of modular knowledge transfer in GPU-optimized reinforcement learning (RL) systems; II) I confirm that environment-specific adaptation is crucial for achieving optimal performance, as evidenced by the slower convergence rates and reduced returns of TL configurations; and III) I provide a comprehensive analysis of how modular TL techniques interact with pure-GPU training frameworks, contributing to the broader understanding of scalable RL architectures. These findings underscore the importance of developing more nuanced task alignment and adaptation strategies to fully realize the potential of TL in RL.

2 Theoretical Background

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a computational approach to learning by interacting with an environment, aiming to achieve long-term goals through trial and error. Unlike supervised learning, which relies on labelled data, RL allows an agent to autonomously explore and learn optimal behaviours by maximizing a numerical reward signal. This interactive process enables RL to address dynamic decision-making tasks in uncertain environments [18].

The foundation of RL lies in the formalism of Markov Decision Processes (MDPs) [1, 10], a mathematical framework that models the interaction between an agent and its environment. An MDP is defined by the tuple (S, A, P, R, γ), where:

- S: A finite set of states representing the environment's possible configurations.
- A: A finite set of actions available to the agent.
- $P(s'|s, a)$: The transition probability of moving to state s' from state s after taking action a.
- $R(s, a)$: The reward function, which maps a state s and an action a to a scalar reward. It defines the environment's mechanism for assigning rewards.
- $\gamma \in [0, 1)$: A discount factor that determines the importance of future rewards.

The objective of RL is to find an optimal policy π^*, a mapping from states to actions, that maximizes the expected cumulative reward or return:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}.$$

This policy guides the agent in balancing exploration (trying new actions to discover their effects) and exploitation (leveraging known actions to maximize rewards) [18]. Specifically, the **behaviour policy** is responsible for this balance, enabling the agent to explore the environment effectively while still exploiting its knowledge to optimize outcomes. In contrast, the **target policy** represents the policy the agent aims to learn and often focuses primarily on exploitation to achieve long-term rewards.

Value functions are central to reinforcement learning, as they quantify the long-term desirability of states under a given policy. The **state-value function**, $v_\pi(s)$, represents the expected return when starting from state s and following policy π thereafter:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right].$$

Here, G_t denotes the cumulative discounted reward starting at time t, and the expectation is taken with respect to the probability distribution induced by policy π.

In addition to $v_\pi(s)$, the **action-value function**, $q_\pi(s, a)$, extends this concept by evaluating specific actions. It quantifies the expected return starting from state s, taking action a, and subsequently following policy π:

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right].$$

Both functions play a pivotal role in RL algorithms, as they enable agents to evaluate and compare states and actions, ultimately guiding the learning and decision-making processes. While $v_\pi(s)$ focuses solely on states, $q_\pi(s, a)$ provides more granular insights by accounting for specific actions, making it particularly useful for action-selection strategies.

These value functions enable efficient policy improvement through iterative methods such as policy iteration and value iteration. Modern RL algorithms build upon these principles to address high-dimensional and complex tasks using function approximation techniques, including neural networks, to represent value functions or policies effectively.

Reinforcement learning provides a powerful framework for solving sequential decision-making problems, making it a cornerstone of modern artificial intelligence research and applications [18].

2.2 Parallelized Q-Networks (PQN)

Parallelized Q-Networks (PQN) based on [20] represent a significant advancement in reinforcement learning by simplifying deep Q-learning while maintaining stability and efficiency. PQN eliminates the need for traditional components like replay buffers and target networks, addressing inherent challenges in stabilizing Temporal Difference (TD) [17] learning with off-policy data and non-linear function approximation.

2.2.1 Motivation and Design

The design of PQN stems from two key observations:

- **Simplification of TD Learning:** Traditional TD algorithms rely heavily on techniques like replay buffers and target networks, which introduce memory overheads and compromise sample efficiency. PQN instead leverages Layer Normalization (LayerNorm) [6] or Batch Normalization (BatchNorm) [4] and synchronous updates across parallel environments to stabilize learning without these additional components.
- **Enhanced Computational Efficiency:** By vectorizing interactions across multiple environments and performing online synchronous updates, PQN achieves a high degree of computational parallelism. This design is particularly suited to GPU-based training pipelines, significantly reducing training time while ensuring stability.

2.2.2 The PQN Algorithm

PQN revisits the classical Q-learning algorithm and incorporates vectorized environments to conduct parallelized interactions and learning. Each batch of interactions is processed simultaneously, with the Q-function parameters updated online using a simple TD update rule:

$$\phi_{i+1} = \phi_i + \alpha_i \Big(r + \gamma \sup_{a'} Q_{\phi_i}(s', a') - Q_{\phi_i}(x) \Big) \nabla_{\phi_i} Q_{\phi_i}(x),$$

where ϕ represents the parameters of the Q-function, α is the learning rate, r is the observed reward, γ is the discount factor, and (s, a, s') represents the state-action-next state transition.

Key features of PQN include:

- **Elimination of Replay Buffers:** Instead of sampling past experiences, PQN relies on real-time interactions with parallel environments, reducing memory requirements and ensuring up-to-date policy evaluations.
- **LayerNorm or BatchNorm Regularization:** LayerNorm or BatchNorm is applied to stabilize the Q-function approximation, mitigating issues related to off-policy instability and non-linear divergence.
- **End-to-End GPU Training:** By avoiding memory-intensive components like replay

buffers, PQN enables fully vectorized training on GPUs, achieving significant speed-ups compared to traditional Deep Q-Network (DQN) implementations.

2.2.3 Empirical Performance and Advantages

PQN has demonstrated competitive performance across various RL benchmarks, including Craftax [9] and multi-agent tasks. Notable benefits include:

- **Faster Convergence:** PQN achieves high performance in fewer gradient updates, demonstrating exceptional sample efficiency.
- **Reduced Complexity:** The algorithm's simplicity, with fewer hyperparameters and straightforward implementation, makes it accessible for a wide range of RL tasks.
- **Scalability:** PQN's compatibility with vectorized environments and end-to-end GPU pipelines enables efficient scaling to complex tasks and larger problem domains.

Overall, PQN redefines the potential of Q-learning by providing a robust, efficient, and scalable alternative to traditional deep RL algorithms, bridging the gap between simplicity and state-of-the-art performance.

2.3 Transfer Learning in Reinforcement Learning

Transfer Learning (TL) is a machine learning paradigm that focuses on leveraging knowledge acquired in one task to improve learning efficiency and performance in a related, yet distinct, target task. In the context of reinforcement learning (RL), transfer learning seeks to reuse policies, features, or representations learned in a source environment to accelerate convergence and enhance performance in a target environment. This approach is particularly appealing given the computational demands of RL, where training an agent from scratch can require substantial time and resources [22].

Reinforcement learning environments often differ in their state space and action space dimensions, posing challenges for knowledge transfer across tasks. For instance, the state space of one environment may be a 10×10 grid, while another environment could represent states in a 15×15 configuration. Similarly, the number of possible actions available to the agent may vary significantly between environments. To address this, the neural network's input and output layers are reinitialized during transfer to match the dimensionalities of the target environment's state and action spaces, while retaining the intermediate layers that encode transferable features.

In this study, transfer learning is applied by first training a network on a source environment, such as Asterix, and then reusing the learned network for a target environment, such as Freeway [21]. To ensure adaptability, the input and output layers of the network are reinitialized to accommodate the state and action spaces of the target task, while the remaining network layers retain the features learned during pretraining. This methodology aligns with prior studies emphasizing the importance of feature transfer and policy reuse in RL [16, 19].

Despite its promise, the efficacy of transfer learning in RL is highly contingent on the similarity between source and target tasks. As highlighted in [19], key factors influencing successful transfer include shared state dynamics, compatible return structures, and overlapping feature representations. However, even with such similarities, challenges such as negative transfer where knowledge from the source task hinders learning in the target task can arise, as demonstrated in studies like [15].

Recent advancements have introduced techniques to mitigate these challenges and enhance transferability. Methods such as policy distillation [11], progressive neural networks [13], and feature extraction have shown promise in improving the alignment between tasks while preserving generalizable knowledge. These approaches provide mechanisms for selectively retaining useful features, avoiding detrimental interference, and adapting to the idiosyncrasies of the target environment. For instance, progressive neural networks use task-specific columns that prevent catastrophic forgetting while allowing shared knowledge across tasks, making them particularly effective in scenarios with incremental task learning [13].

The experimental setup in this study adopts a practical transfer learning framework, reinitializing key layers to handle variations in state and action

spaces while evaluating whether shared features can accelerate learning in the target task. However, as observed in prior work, the limitations of benchmark environments such as MinAtar [21] with its simplified dynamics and low-dimensional state representations may reduce the richness of transferable features, posing additional constraints on the effectiveness of transfer learning.

Ultimately, transfer learning in RL remains a promising yet complex area of study. While it holds the potential to improve sample efficiency and performance in computationally intensive tasks, its success relies on algorithmic advancements, carefully curated source-target pairs, and robust evaluation frameworks that can better capture the nuances of real-world applications [3, 7, 22].

3 Methods

3.1 Objective

The objective of this research is to evaluate whether TL can be achieved in RL using PQN. TL is particularly relevant for PQN because it enables the reuse of knowledge from previously learned tasks, potentially reducing the computational effort required to train a model in a new but related environment. By leveraging shared features across tasks, TL aims to accelerate the learning process, reduce convergence times, and improve overall training efficiency. The primary research question is: *What impact does Transfer Learning have on the learning time of Parallelized Q-Network in the MinAtar environment?*

3.2 Model and Algorithms

The experiments in this study utilize the foundational PQN implementation combined with a convolutional neural network (CNN) architecture. The PQN framework is specifically designed to maximize training efficiency by leveraging GPU-based parallelization, allowing for the simultaneous handling of multiple tasks. The CNN serves as the backbone of the model, enabling it to extract spatial features from the MinAtar games, which are represented as compact, low-dimensional grids. This setup aligns with the computational constraints of RL tasks in minimalist environments like MinAtar.

The CNN begins with a convolutional layer that uses 16 filters, each with a kernel size of 3×3, a stride of 1, and "valid" padding, ensuring no unnecessary spatial information is added. The kernel weights are initialized using the He normal initialization method to ensure optimal convergence. This convolutional layer is followed by a normalization step, which can employ either LayerNorm or BatchNorm, depending on the configuration. Layer normalization ensures stable gradients by normalizing across features within a layer, while batch normalization adjusts the input distribution dynamically during training.

The output of the convolutional layer is passed through a ReLU activation function, introducing non-linearity and enabling the network to model complex functions. The resulting tensor is then flattened into a one-dimensional vector to prepare it for the fully connected layers. The first dense layer reduces the high-dimensional representation to a latent space of 128 neurons, using He normal initialization for weight optimization. Once again, a normalization step and ReLU activation are applied to refine the latent representation.

The final dense layer maps the latent representation to the action space of the environment, producing Q-values for each possible action. These Q-values represent the expected cumulative returns associated with taking a particular action in a given state, guiding the agent's decision-making process.

Overall, the neural network structure consists of a convolutional layer for feature extraction, followed by a series of dense layers for representation learning and action-value computation. This modular design ensures computational efficiency while maintaining the flexibility to adapt to the specific requirements of various MinAtar environments.

3.3 Environments and Tasks

The experiments focus on four games from the MinAtar environment: *Asterix*, *SpaceInvaders*, *Freeway*, and *Breakout*. These games were chosen due to their inclusion in the baseline benchmark of the PQN, ensuring comparability with existing results. Each game presents unique challenges and patterns, ranging from object collection in *Asterix* to avoiding obstacles in *Freeway*, making them suitable for evaluating the generalization capabilities of TL.

3.4 Experimental Metrics and Variables

The primary metric of interest is the learning time or convergence time, defined as the time taken for the return to plateau over training steps. This metric serves as a proxy for computational efficiency and reflects how quickly the agent adapts to the environment. The experiments evaluate TL configurations to determine whether knowledge learned in one environment can expedite training in a similar environment. Here, similarity between environments is defined based on the shared input feature space, action space, and overall task structure. For instance, *Asterix* and *Breakout* share identical input layer sizes and have similar visual representations, making them ideal candidates for evaluating TL effectiveness. For each configuration, returns are recorded as a function of the number of environment steps, enabling a direct comparison of convergence behavior.

3.5 Configurations

Two transfer learning configurations are compared in this study:

1. **Output-Layer-Only Reinitialization**: This configuration is applied to *Asterix* and *Breakout*, which share the same input layer size. It assumes that task-specific features reside primarily in the output layer, allowing the input features to be reused across tasks. We will refer to this configuration as *Output-Reinit* throughout the article.

2. **Input-Output-Layer Reinitialization**: This configuration is applied to all other cases where both the input and output layers are reinitialized. It assumes that substantial task-specific adaptations are required in both the input and output layers to accommodate different state representations and action spaces. We will refer to this configuration as *In-Out-Reinit* throughout the article.

3.6 Baselines and Experimental Setup

The baseline runs follow the default settings provided by the PQN developers. The only modification is an increase in environment steps from 10 million to 20 million to ensure a clear visualization of the learning speed. Each configuration, including the baseline, is run for 20 million environment steps, with ten runs per configuration. This adjustment ensures sufficient training time to observe meaningful trends in return progression and convergence.

3.7 Implementation Details

The experiments are implemented in Python using the following libraries and tools:

- **JAX, Gymnax, Flax:** These libraries form the core framework for building and training the PQN model.
- **Hydra:** Used for configuration management, enabling easy switching between experimental setups.
- **Wandb:** Facilitates fast visualization and tracking of training progress.
- **NumPy, Pandas, Seaborn, Matplotlib:** Used for data analysis and creating publication-quality visualizations.

Several small shell scripts are used to automate multiple runs for each configuration, streamlining the execution process and reducing manual intervention.

3.8 Hardware Specifications

The experiments are conducted on a personal computer equipped with an NVIDIA RTX 3060 Mobile GPU and an AMD Ryzen 7 5000 series CPU. This hardware setup balances computational efficiency with accessibility, ensuring that the findings can be replicated on standard consumer-grade devices.

3.9 Data Collection and Analysis

Returns are collected at each environment step, providing a detailed measure of the agent's performance over time. For each configuration, the mean and standard error of the returns are calculated across 10 independent runs, ensuring statistical reliability and minimizing the influence of outliers. The final plots illustrate the relationship

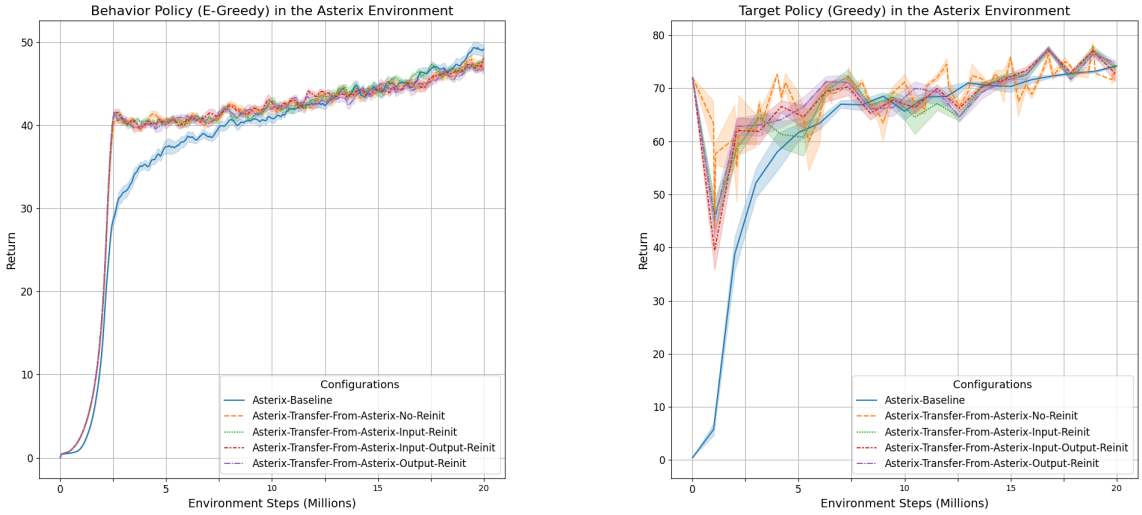

Figure 3.1: Validation of the neural network loading mechanism in the Asterix environment. Left: Behavior Policy learning curve. Right: Target Policy learning curve. Both curves confirm that the loaded network reproduces its original performance.

between return and environment steps, offering a visual comparison of the convergence rates, stability, and overall learning performance of the various configurations. These visualizations serve as a crucial tool for analyzing the impact of transfer learning strategies, shedding light on their ability to accelerate the learning process, improve stability, and achieve higher returns within a given environment.

3.10 Reproducibility

The experiments are reproducible using the code and data provided in the publicly accessible GitHub repository. Random seeds are explicitly set in the `config.yaml` file (**NUM_SEEDS** 1 and **SEED** 0) to ensure consistent results across runs. Detailed instructions for setting up the experimental environment, including Docker images and shell scripts, are included in the repository. Additionally, a structured README file in the repository provides an overview of the project organization, including descriptions of the `purejaxql`, `docker`, and `results` directories to assist reproduction efforts. The current implementation relies on a Docker container to resolve dependency conflicts, ensuring consistency across environments. Future improvements could include enhancing flexibility by enabling execution outside of Docker.

3.11 Mehtod Validation

An additional validation step was performed to ensure the correctness of the neural network loading procedure. In this step, the pre-trained network was deployed in the same environment it was originally trained on. This validation confirmed that the network's parameters were correctly preserved and that the loading mechanism did not inadvertently introduce inconsistencies.

Figure 3.1 illustrates the learning curves for the behaviour and target policies, respectively, in the Asterix environment. The results demonstrate that the pre-trained network successfully replicates its previously achieved performance, further verifying the integrity of the model loading process. This step was critical to eliminate potential errors in the experimental setup and ensure that subsequent experiments involving transfer learning were based on a reliable baseline configuration.

Beyond validation, this experiment also serves to illustrate the direct effects of reinitialization on learning dynamics. Since the transfer occurs within

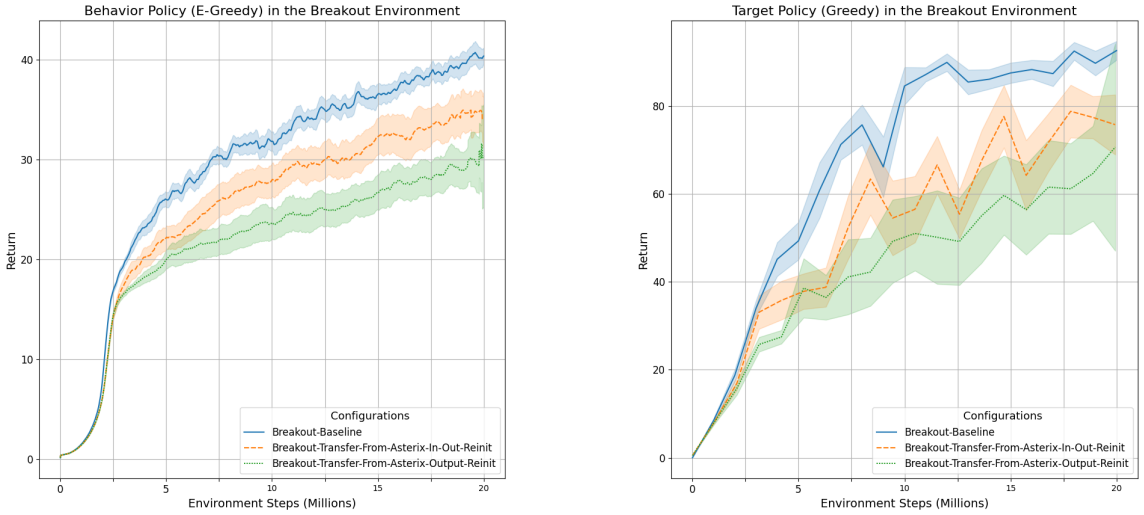

Figure 4.1: Behavior and Target Policy learning curves for the Breakout environment using the Output-Reinit and In-Out-Reinit configurations compared to the baseline.

the same environment, it provides insight into how much information is retained when only the output, only the input, or both layers are reinitialized. However, because the source and target environments are identical, this experiment does not directly test the effectiveness of reinitialization in transfer learning across different environments. Instead, it isolates the impact of reinitialization without additional confounding factors, such as differences in task structure or state-action representations.

3.12 Challenges and Limitations

Ensuring the transfer process was functioning correctly posed a significant challenge. To verify the transfer code, a network was trained on an environment and then reused in the same environment. If the target policy graph started at the plateau value recorded during training, it confirmed that the network was being loaded correctly. This validation process was critical to ensure the theoretical feasibility of testing TL across all environment combinations. Once confirmed, the TL experiments could proceed with confidence in the implementation's correctness.

Currently, the code is only capable of running inside the specifically crafted Docker container provided by the developers. While this ensures consistency and eliminates dependency conflicts, it also limits flexibility. Future work could focus on making the dependencies more flexible, allowing the code to be run outside the Docker container or in diverse computational environments.

4 Results

4.1 Output-Reinit vs In-Out-Reinit

Figures 4.1 and 4.2 illustrate the learning outcomes of PQN agents trained on Asterix and Breakout under different transfer learning configurations. The results evaluate baseline training, the Output-Reinit configuration, and the In-Out-Reinit configuration.

Breakout Environment

- **Baseline configuration**: The baseline continues to achieve the highest target policy mean return (92.56 ± 2.02) and behaviour policy mean return (42.03 ± 1.03). These results affirm the stability and effectiveness of task-specific learning without transfer learning.

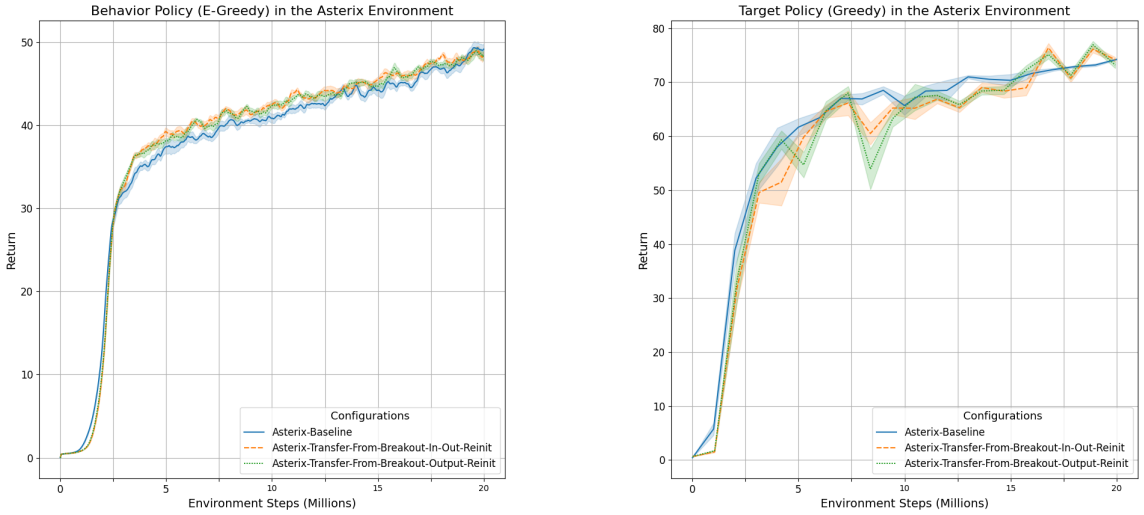

Figure 4.2: Behavior and Target Policy learning curves for the Asterix environment using the Output-Reinit and In-Out-Reinit configurations compared to the baseline.

- **Output-Reinit configuration**: This configuration shows a considerable reduction in returns compared to the baseline, achieving a target policy mean the return of (62.03 ± 8.56) and a behaviour policy mean the return of (29.00 ± 2.11). The significantly slower convergence and lower returns reflect limited benefits from reusing pre-trained features while reinitializing only the output layer.

- **In-Out-Reinit configuration**: This configuration exhibits the lowest returns overall, with a target policy mean the return of (74.86 ± 5.07) and a behaviour policy mean the return of (34.67 ± 1.80). This suggests that fully reinitializing the network results in pronounced negative transfer effects, hindering performance.

Figure 4.1 illustrates that neither transfer configuration surpasses the baseline in terms of learning speed or returns. The baseline demonstrates rapid convergence and consistently superior performance throughout the training process. By contrast, the Output-Reinit and In-Out-Reinit configurations show slower learning and fail to leverage any meaningful advantage from pre-trained features. The marked reductions in both target and behavior policy returns for the transfer configurations underscore the inefficacy of the transfer approaches in this environment. These findings strongly suggest that in the Breakout environment, environment-specific training continues to be the most effective strategy, with transfer learning introducing more challenges than benefits.

Table A.1 provides a summary of the mean return values for the Breakout environment, further emphasizing the baseline's superior performance and the inefficiencies introduced by transfer configurations. The marked reductions in both target and behaviour policy returns for the transfer configurations underscore the inefficacy of the transfer approaches in this environment. These findings strongly suggest that in the Breakout environment, environment-specific training continues to be the most effective strategy, with transfer learning introducing more challenges than benefits.

Asterix Environment

- **Baseline configuration**: The baseline configuration achieves the highest target policy mean return (74.22 ± 0.08) and behaviour policy mean return (49.89 ± 0.71), serving as the

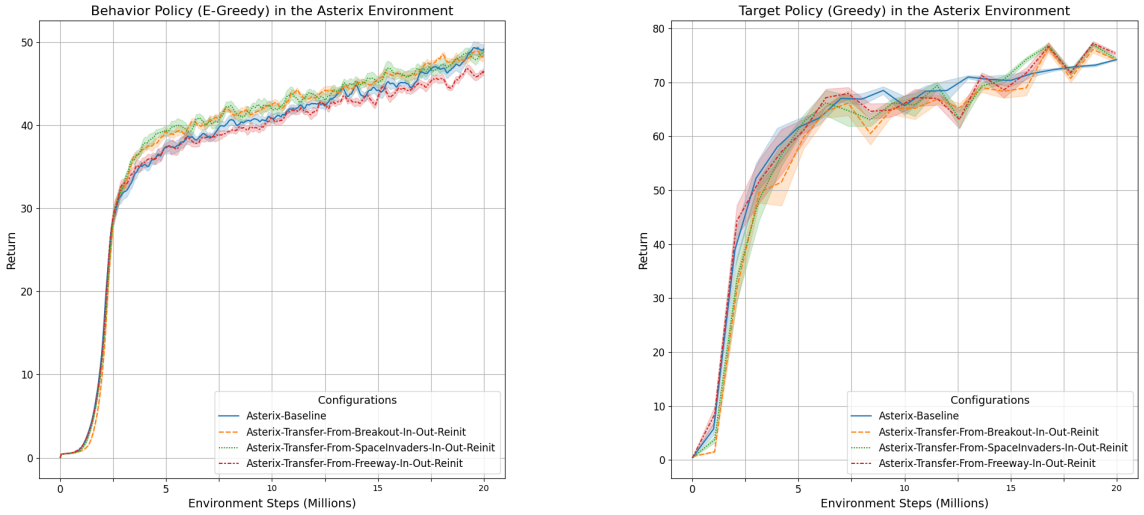

Figure 4.3: Behavior and Target Policy learning curves for the Asterix environment, comparing the Baseline configuration with In-Out-Reinit transfers from Breakout, Freeway, and Space Invaders. The left panel presents the Behavior Policy results, while the right panel displays the Target Policy performance.

benchmark for evaluating transfer configurations.

- **Output-Reinit configuration**: This configuration shows slightly lower returns compared to the baseline, achieving a target policy mean the return of (73.81 ± 0.78) and a behaviour policy mean the return of (49.80 ± 0.64). The results highlight its inability to improve learning speed or achieve higher returns.
- **In-Out-Reinit configuration**: This configuration attains similar behavior policy returns (48.78 ± 0.95) and a slightly higher target policy mean return of (74.27 ± 0.42). Despite these results, it fails to surpass the baseline in terms of learning speed or overall effectiveness.

Figure 4.2 demonstrates that neither transfer configuration improves learning speed in the Asterix environment. Both configurations exhibit slower initial progress compared to the baseline, with the Output-Reinit configuration particularly struggling to converge efficiently. Although the returns for both transfer configurations are relatively close to one another, they fail to match the baseline's performance. These results suggest that knowledge transfer between tasks in this setup is limited. Despite retaining some pre-trained features, the transfer configurations do not accelerate learning or provide meaningful gains in this environment.

Table 4.1 summarizes the mean return values for the Asterix environment, highlighting the baseline configuration's superior performance. These results suggest that knowledge transfer between tasks in this setup is limited. Despite retaining some pre-trained features, the transfer configurations do not accelerate learning or provide meaningful gains in this environment.

4.2 Overall Comparison

In this section, we analyze the learning behaviour of all configurations compared to the baseline across four distinct environments: **Asterix**, **Breakout**, **Freeway**, and **SpaceInvaders**. Each configuration employs the In-Out-Reinit approach, where both the input and output layers are reinitialized during transfer. This methodology ensures that task-specific features are not retained

Table 4.1: Asterix Environment Means

Configuration	Target Policy Mean ± SE	Behavior Policy Mean ± SE
Asterix-Baseline	74.22 ± 0.08	49.89 ± 0.71
Asterix-Transfer-From-Breakout-Output-Reinit	73.81 ± 0.78	49.80 ± 0.64
Asterix-Transfer-From-Breakout-In-Out-Reinit	74.27 ± 0.42	48.78 ± 0.95
Asterix-Transfer-From-SpaceInvaders-In-Out-Reinit	74.47 ± 0.35	49.58 ± 0.95
Asterix-Transfer-From-Freeway-In-Out-Reinit	75.76 ± 0.35	47.54 ± 0.84

across tasks, providing a consistent framework for evaluating the impact of knowledge transfer. Notably, the exclusive use of the In-Out-Reinit approach eliminates any potential variability that might arise from alternative reinitialization strategies, ensuring uniformity in our analysis. Despite the theoretical potential of transfer learning, the results consistently demonstrate that baseline training outperforms all transfer configurations in both learning speed and convergence characteristics. Below, we provide a detailed discussion for each environment, emphasizing the observed limitations of transfer learning under these conditions.

Asterix Environment:

- **Asterix-Baseline:**
  Attains the highest Target Policy mean return of (74.22 ± 0.08) and Behavior Policy mean return of (49.89 ± 0.71). This configuration sets the benchmark for evaluating learning efficiency in the Asterix environment.

- **Asterix-Transfer-From-Breakout:**
  Produces a Target Policy mean return of (74.27 ± 0.42) and a Behavior Policy mean return of (48.78±0.95), demonstrating comparable returns but slower adaptation despite leveraging knowledge from Breakout.

- **Asterix-Transfer-From-Freeway:**
  Reports a Target Policy mean return of (75.76 ± 0.35) and a Behavior Policy mean return of (47.54 ± 0.84). Although achieving higher Target Policy returns, its Behavior Policy performance is less consistent than the baseline.

- **Asterix-Transfer-From-SpaceInvaders:**
  Yields a Target Policy mean return of (74.47 ± 0.35) and a Behavior Policy mean return of (49.58 ± 0.95). Although this configuration approaches baseline returns, it demonstrates slower learning progress.

Figure 4.3 shows the learning curves for the Behavior and Target policies, respectively, highlighting the superior convergence speed of the baseline configuration. The mean return values for each configuration are summarized in Table 4.1, reinforcing the observed trends. These findings emphasize the necessity of task-specific learning to account for unique state-action dynamics in the Asterix environment.

For the remaining environments:

- **Breakout Environment:** The baseline demonstrates the highest Behavior Policy performance (42.03 ± 1.03), while transfer configurations show slower convergence and underperform in Behavior Policy returns.

- **Freeway Environment:** The baseline achieves the best overall results, with transfer configurations exhibiting delayed learning and failing to match the baseline's returns.

- **Space Invaders Environment:** Although some transfer configurations achieve competitive Target Policy returns, the baseline consistently outperforms in terms of both learning speed and final returns.

Detailed results for these environments are available in Appendix A (mean tables) and Appendix B (learning curves).

5 Conclusion and Discussion

While transfer learning has demonstrated significant success in fields such as computer vision and natural language processing, its impact in reinforcement learning has been comparatively limited. Unlike these domains, where pretraining on large datasets often leads to substantial improvements, RL involves dynamically interacting with environments, making the transfer of learned policies or representations more complex. Differences in reward structures, action spaces, or task dynamics between source and target environments can hinder effective transfer, often leading to a phenomenon known as negative transfer, where prior knowledge from the source task impedes rather than accelerates learning in the target task. This inherent complexity has made it challenging to replicate the transformative effects of transfer learning seen in other machine learning fields.

This observation raises a critical question: why does transfer learning, which has proven effective in other machine learning domains, yet fail to provide significant benefits in this context? To answer this, we delve into the challenges of applying transfer learning, the limitations of the MinAtar environment, and the potential factors contributing to the observed underperformance. By analyzing these issues, we aim to highlight broader implications for future research in transfer learning for reinforcement learning systems.

5.1 Challenges with Transfer Learning

A key challenge in leveraging transfer learning in reinforcement learning is balancing the integration of knowledge from prior tasks with the dynamic, task-specific demands of the target environment. While PQN demonstrates some capacity for multitask learning, the results of this study suggest that its multitask capabilities may not be fully optimized for effective transfer learning. Although PQN can handle multiple tasks to an extent, the mechanisms it employs for knowledge sharing and adaptation are not as robust as those seen in dedicated multitask frameworks. This limitation may have contributed to the lack of gains observed during the stages of training for transfer configurations.

One possible explanation lies in how PQN processes task-specific and transferable information. Unlike architectures explicitly designed for transfer, PQN lacks mechanisms to prioritize or selectively refine useful features from the source task. Consequently, during the initial stages of training, the network may inadvertently overwrite or dilute pretrained knowledge as it aggressively explores the target environment. This tendency can reduce the utility of transfer, particularly when the source and target tasks share only limited structural similarities.

Moreover, while PQN's multitask capabilities enable it to learn across tasks to some degree, its design does not explicitly account for the challenges of negative transfer. For example, in this study, all transfer configurations exhibited slower convergence compared to the baseline, highlighting instances where pretrained features from the source task may have conflicted with the optimal learning trajectory of the target task. This observation underscores the need for more sophisticated task-alignment mechanisms that can distinguish between transferable and non-transferable features.

Ultimately, the findings suggest that while PQN has the potential to benefit from transfer learning, achieving significant improvements requires a combination of task-specific adjustments and algorithmic enhancements. Future research could explore augmenting PQN with selective feature extraction techniques to improve its capacity to handle more complex transfer scenarios. These insights emphasize that effective transfer learning in RL demands both algorithmic and task-specific considerations, even when the underlying model demonstrates multitask learning capabilities.

5.2 Limitations of the MinAtar Environment

The MinAtar environment, while a valuable tool for benchmarking RL algorithms, introduces significant limitations that may impact the effectiveness of transfer learning. Designed as a simplified version of Atari environments, MinAtar reduces visual complexity by employing compact, low-dimensional state representations and deterministic dynamics. While this design makes it computationally efficient and easier to evaluate algorithms, it may inadvertently diminish the potential for meaningful transfer learning by simplifying the relationships

between states, actions, and rewards. The limited task diversity within MinAtar further restricts the availability of transferable features, as the tasks often lack the nuanced complexities seen in larger-scale benchmarks.

For instance, in this study, configurations pre-trained on tasks such as Asterix or Breakout often failed to produce significant gains when applied to Freeway or Space Invaders. A plausible explanation lies in the simplified nature of MinAtar tasks, where the differences between source and target tasks may not be rich or diverse enough to warrant the application of transfer learning. Without the presence of complex visual patterns, intricate dynamics, or high-dimensional representations, the utility of transferring learned features is greatly reduced, as many of these features may already be trivial to learn from scratch in the target task.

Furthermore, MinAtar's lack of stochasticity and limited task granularity may have exacerbated the challenges of transfer learning observed in this study. In more complex environments, such as those found in the Atari Learning Environment (ALE) [7] or real-world RL domains, tasks often require learning hierarchical policies or abstract representations that can be shared across multiple environments. In contrast, MinAtar's simplicity reduces the need for such shared abstractions, making transfer learning less impactful.

Given these limitations, future studies could benefit from exploring transfer learning in richer environments that provide greater task diversity and complexity. Benchmarks such as ALE [7], Maniskill2 [3], or IssacGym [8] may offer a more challenging and realistic testbed for evaluating the potential of transfer learning. By incorporating high-dimensional observations, stochastic dynamics, and diverse task requirements, such environments could provide a clearer picture of how transfer learning techniques scale to real-world scenarios and complex RL tasks.

5.3 Future Work

The findings of this study highlight the limitations of transfer learning in the MinAtar environment and suggest several promising directions for future research. First, exploring hierarchical techniques could significantly enhance transferability across tasks. Hierarchical learning frameworks can enable agents to decompose complex tasks into reusable sub-components, fostering more effective knowledge sharing. Similarly, meta-learning approaches, as discussed by [22], can improve the adaptability of RL algorithms by teaching them to learn general strategies that extend beyond individual tasks. These methods could help mitigate the negative transfer observed in this study by focusing on transferable abstractions rather than task-specific features.

Second, testing transfer learning strategies in richer and more diverse benchmark environments is essential. MinAtar's simplicity may have limited the scope of transferable features, underscoring the need for environments with greater task diversity and representational complexity. Platforms such as the ALE [7], IsaacGym [8], and ManiSkill2 [3] offer more realistic and stochastic environments that could better demonstrate the potential of transfer learning techniques. As noted in [19], increasing the representational richness and diversity of tasks is key to understanding how transfer learning can scale to more complex RL scenarios.

Third, future research should examine advanced transfer learning strategies tailored to reinforcement learning domains. In [19], authors identified methods such as instance transfer, feature transfer, and policy transfer as critical avenues for improving RL performance across tasks. Building on these ideas, recent advances like [12] and feature extraction can further refine how knowledge is reused between source and target tasks. Additionally, studies like [15] emphasize the importance of ensuring task alignment to avoid negative transfer, which was evident in some configurations of this study. By combining these insights, future efforts could develop robust algorithms capable of dynamically adapting transferred knowledge to maximize learning efficiency.

Lastly, integrating adaptive exploration techniques with transfer learning mechanisms could address challenges related to overwriting or destabilizing pre-trained knowledge. RL agents often rely on aggressive exploration strategies like epsilon greedy that can interfere with the reuse of learned features. By incorporating mechanisms that balance exploration with the preservation of useful knowledge, it may be possible to enhance both the speed and quality of learning in transfer scenarios.

These directions highlight the need for further

research into the intersection of algorithm design, task complexity, and knowledge transfer. Advancing transfer learning for RL will be critical for scaling algorithms to real-world applications, where leveraging prior knowledge is essential for efficient and robust decision-making.

5.4 Final Conclusion

This study provides a detailed exploration of the effectiveness of transfer learning within reinforcement learning domains, using the Parallelized Q-Network and the MinAtar environment as a controlled experimental framework. The findings reveal that while transfer learning offers modest advantages during the initial stages of training, its impact diminishes as task-specific learning dynamics dominate. This underscores the inherent complexity of transfer learning in RL, where the intricate interplay between algorithm design, task structure, and transferable features significantly influences outcomes.

The limited performance improvements observed in this work highlight the importance of carefully aligning source and target tasks to ensure meaningful transfer. While computationally efficient environments such as MinAtar may lack the task diversity and representational richness required to fully exploit transfer learning techniques, other factors may also contribute to the observed limitations. For instance, prior work by Sabatelli et al. [15] attributes poor transfer performance to the overfitting of the target network. However, this explanation does not fully align with the results of this study, as PQN does not employ a separate target network, yet negative transfer remains prevalent. This suggests that the root cause may instead lie in the structural properties of the underlying Markov Decision Processes (MDPs), as discussed by Taylor and Stone [19]. Differences in state spaces, action spaces, and reward functions between source and target tasks could impede effective knowledge transfer, even in algorithms that aim to generalize across tasks.

Ultimately, this study reinforces the nuanced role of transfer learning in RL, demonstrating that its efficiency depends not only on the underlying algorithm but also on the interplay between the environment, task dynamics, and the mechanisms used to facilitate the transfer. Advancing the field will require continued efforts to design adaptive algorithms, identify task-alignment strategies, and develop benchmarks that reflect the challenges of real-world decision-making scenarios. By addressing these areas, transfer learning can become a cornerstone for scaling RL to increasingly complex and practical applications.

References

[1] R. Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

[2] M. Gallici, M. Fellows, B. Ellis, B. Pou, I. Masmitja, J. N. Foerster, and M. Martin. Simplifying deep temporal difference learning. *arXiv preprint arXiv:2407.04811*, 2024.

[3] J. Gu, F. Xiang, X. Li, Z. Ling, X. Liu, T. Mu, Y. Tang, S. Tao, X. Wei, Y. Yao, et al. Maniskill2: A unified benchmark for generalizable manipulation skills. *arXiv preprint arXiv:2302.04659*, 2023.

[4] S. Ioffe. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[5] R. T. Lange. gymnax: A jax-based reinforcement learning environment library, 2022. *URL http://github. com/RobertTLange/gymnax*, 2 (3):4, 2022.

[6] J. Lei Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *ArXiv e-prints*, pages arXiv–1607, 2016.

[7] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. J. Hausknecht, and M. Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

[8] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

[9] M. Matthews, M. Beukman, B. Ellis, M. Samvelyan, M. Jackson, S. Coward, and J. Foerster. Craftax: A lightning-fast benchmark for open-ended reinforcement learning. *arXiv preprint arXiv:2402.16801*, 2024.

[10] M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[11] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

[12] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

[13] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[14] A. Rutherford, B. Ellis, M. Gallici, J. Cook, A. Lupu, G. Ingvarsson, T. Willi, A. Khan, C. S. de Witt, A. Souly, et al. Jaxmarl: Multi-agent rl environments in jax. *arXiv preprint arXiv:2311.10090*, 2023.

[15] M. Sabatelli and P. Geurts. On the transferability of deep-q networks. *arXiv preprint arXiv:2110.02639*, 2021.

[16] R. Sasso, M. Sabatelli, and M. A. Wiering. Multi-source transfer learning for deep model-based reinforcement learning. *arXiv preprint arXiv:2205.14410*, 2022.

[17] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44, 1988.

[18] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[19] M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009.

[20] C. J. C. H. Watkins. Learning from delayed rewards. 1989.

[21] K. Young and T. Tian. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. *arXiv preprint arXiv:1903.03176*, 2019.

[22] Z. Zhu, K. Lin, A. K. Jain, and J. Zhou. Transfer learning in deep reinforcement learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

A Mean return tables

Table A.1: Breakout Environment Means

Configuration	Target Policy Mean ± SE	Behavior Policy Mean ± SE
Breakout-Baseline	92.56 ± 2.02	42.03 ± 1.03
Breakout-Transfer-From-Asterix-In-Out-Reinit	74.86 ± 5.07	34.67 ± 1.80
Breakout-Transfer-From-Freeway-In-Out-Reinit	95.03 ± 1.05	38.97 ± 1.35
Breakout-Transfer-From-SpaceInvaders-In-Out-Reinit	88.13 ± 7.01	37.14 ± 1.64
Breakout-Transfer-From-Asterix-Output-Reinit	62.03 ± 8.56	29.00 ± 2.11

Table A.2: Freeway Environment Means

Configuration	Target Policy Mean ± SE	Behavior Policy Mean ± SE
Freeway-Baseline	67.11 ± 0.14	60.53 ± 0.10
Freeway-Transfer-From-SpaceInvaders-In-Out-Reinit	66.61 ± 0.25	59.76 ± 0.14
Freeway-Transfer-From-Asterix-In-Out-Reinit	66.56 ± 0.27	59.74 ± 0.12
Freeway-Transfer-From-Breakout-In-Out-Reinit	65.23 ± 0.62	59.38 ± 0.11

Table A.3: Space Invaders Environment Means

Configuration	Target Policy Mean ± SE	Behavior Policy Mean ± SE
SpaceInvaders-Baseline	161.70 ± 0.97	117.63 ± 1.21
SpaceInvaders-Transfer-From-Freeway-In-Out-Reinit	161.60 ± 1.06	110.46 ± 1.70
SpaceInvaders-Transfer-From-Asterix-In-Out-Reinit	158.70 ± 2.49	113.39 ± 2.41
SpaceInvaders-Transfer-From-Breakout-In-Out-Reinit	162.80 ± 1.24	111.26 ± 1.63

B Supplementary Figures

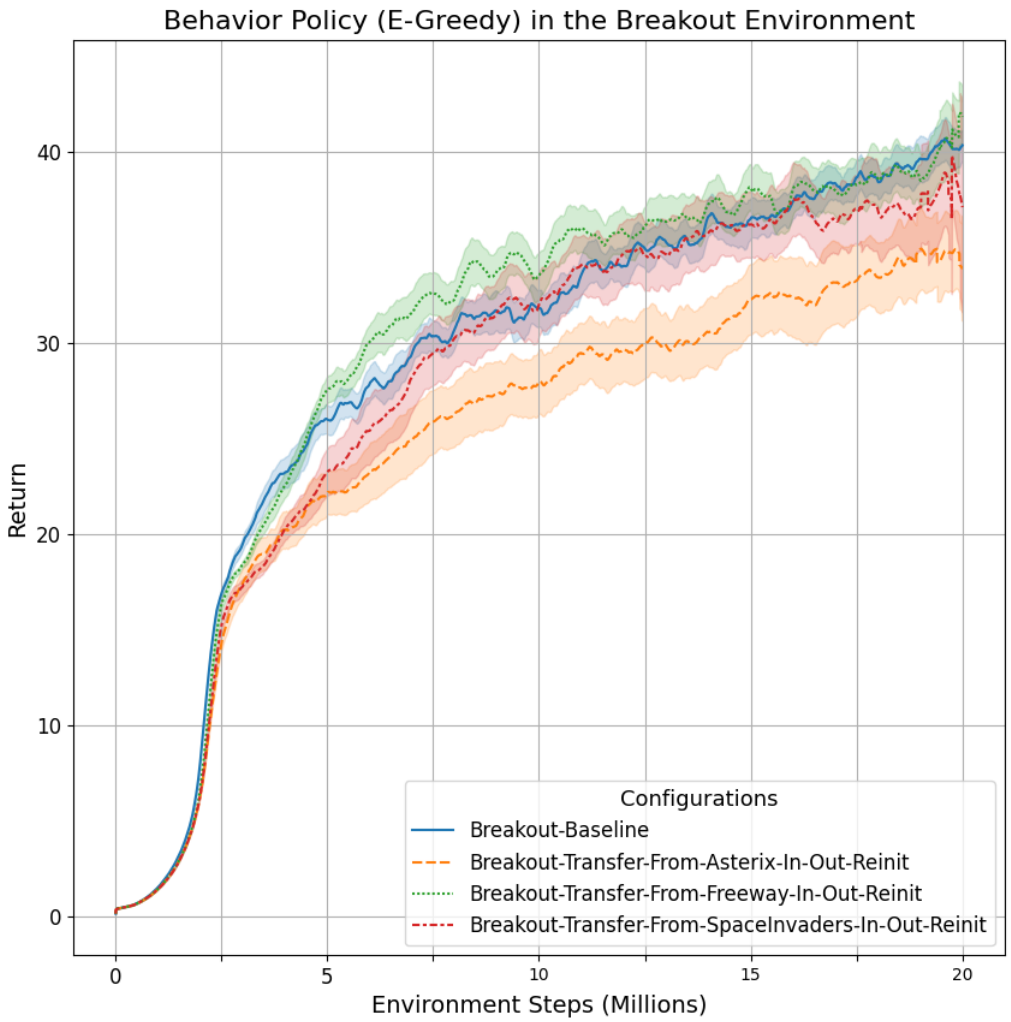

Figure B.1: Behavior Policy learning curves in the Breakout environment for various transfer configurations. Shown here are the average returns over 20 million steps.

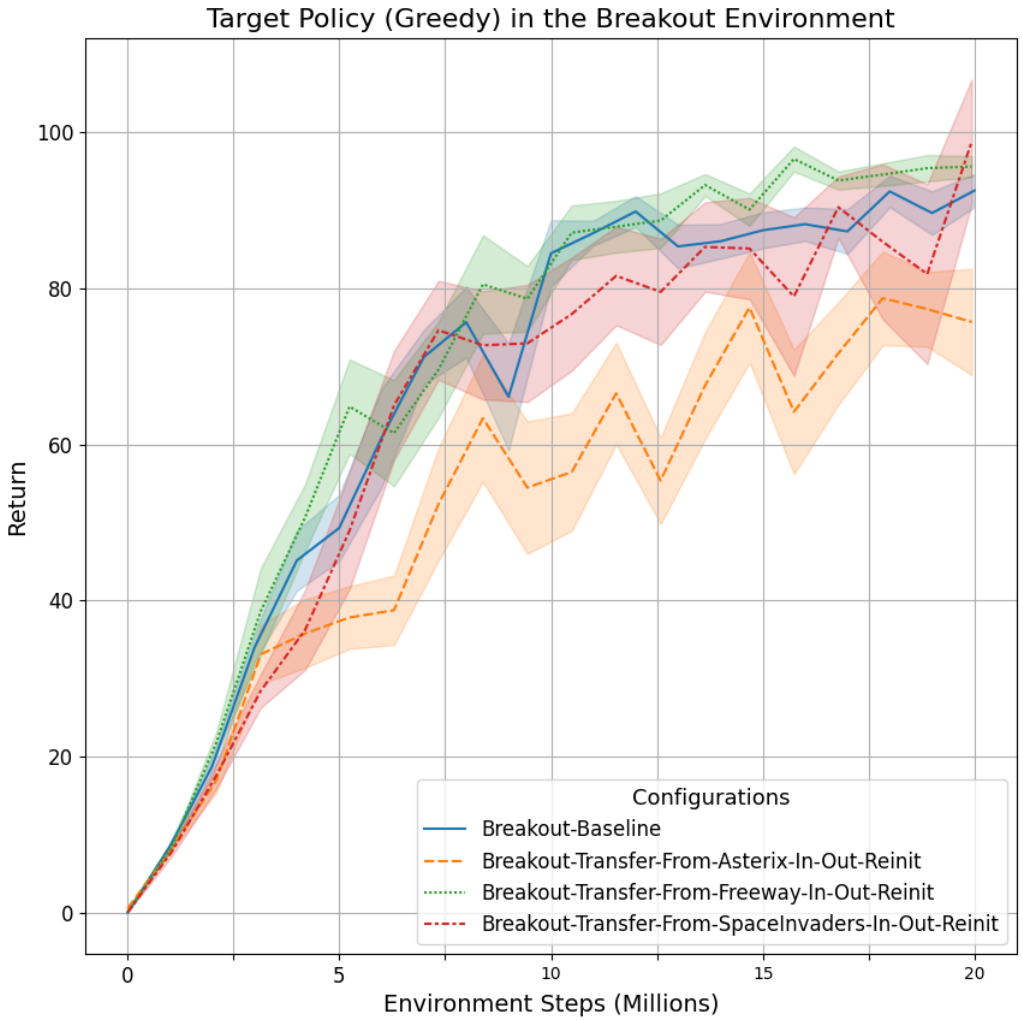

Figure B.2: Target Policy learning curves in the Breakout environment for various transfer configurations. The training was performed for 20 million steps, illustrating the evolution of mean returns.

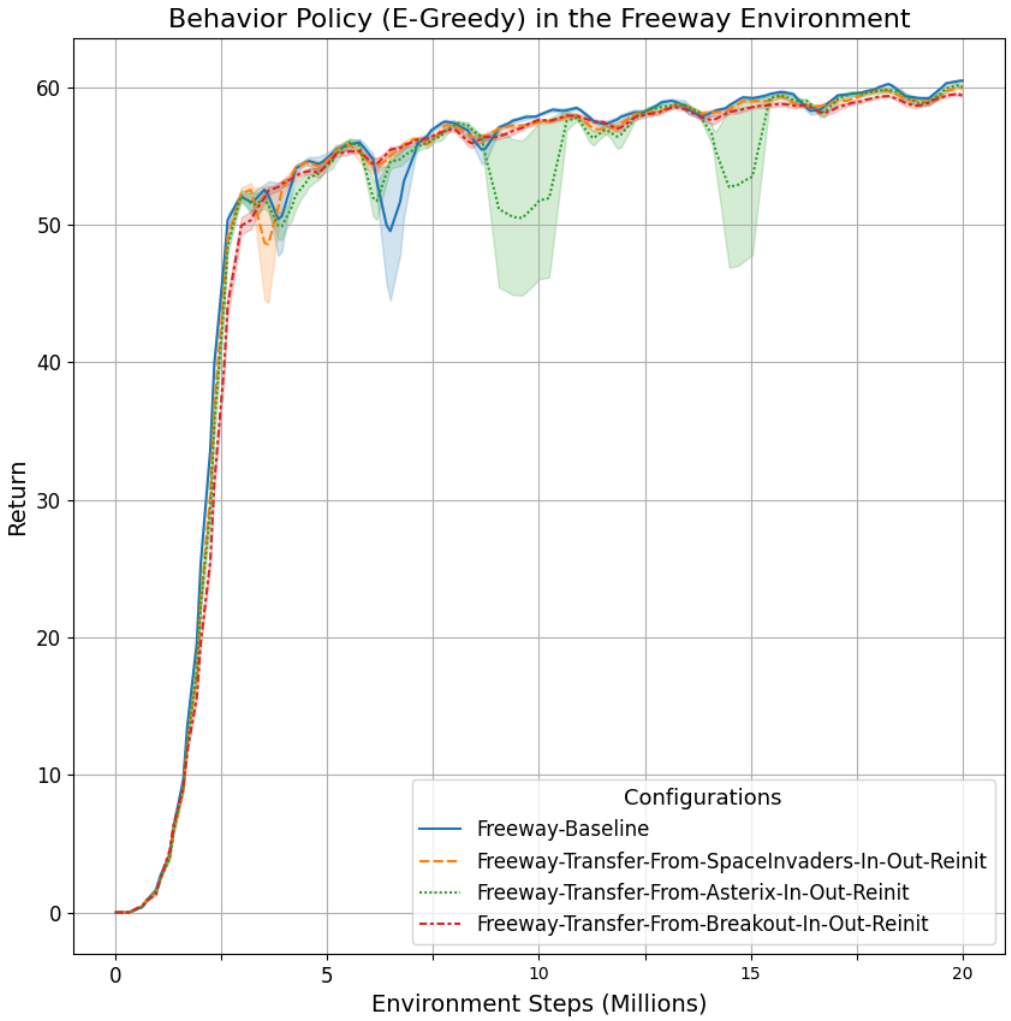

Figure B.3: Behavior Policy learning curves in the Freeway environment for various transfer configurations. Shown here are the average returns over 20 million steps.

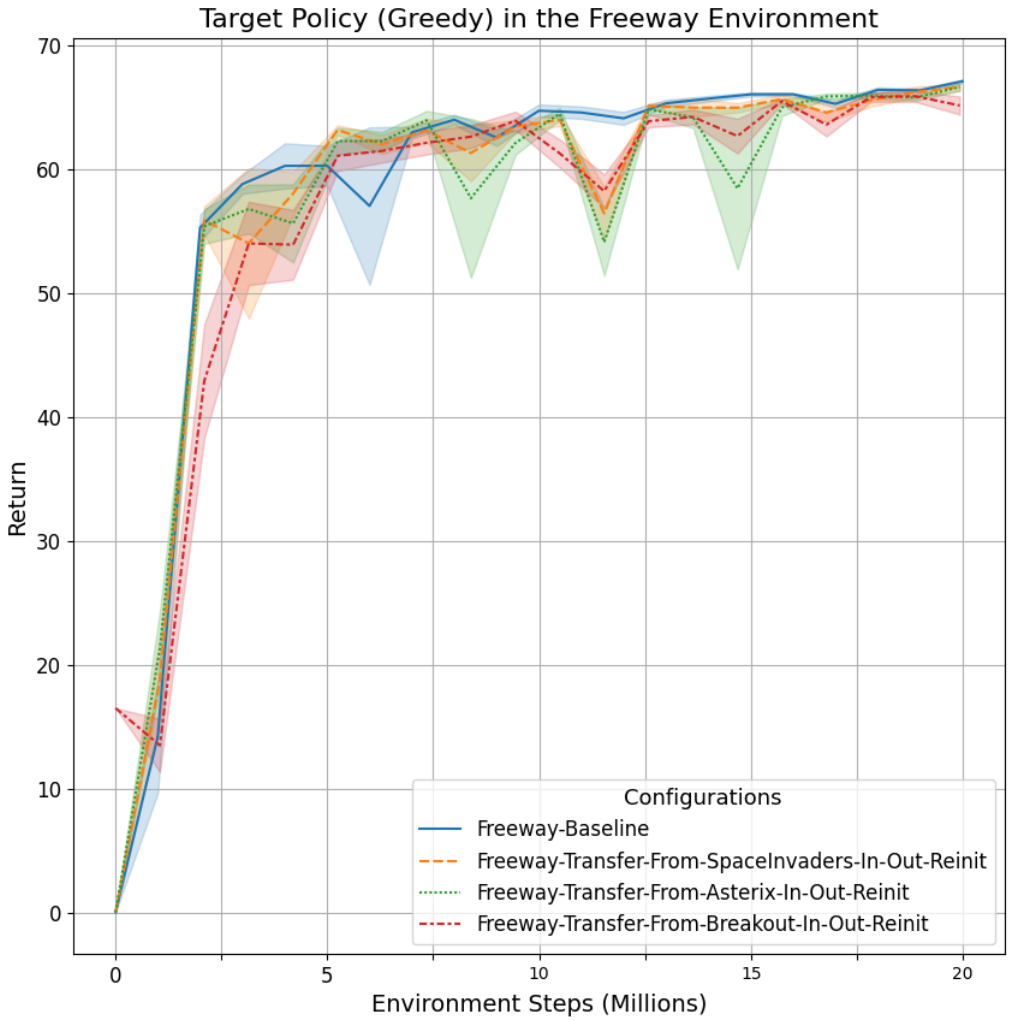

Figure B.4: Target Policy learning curves in the Freeway environment for various transfer configurations. The training was performed for 20 million steps, illustrating the evolution of mean returns.

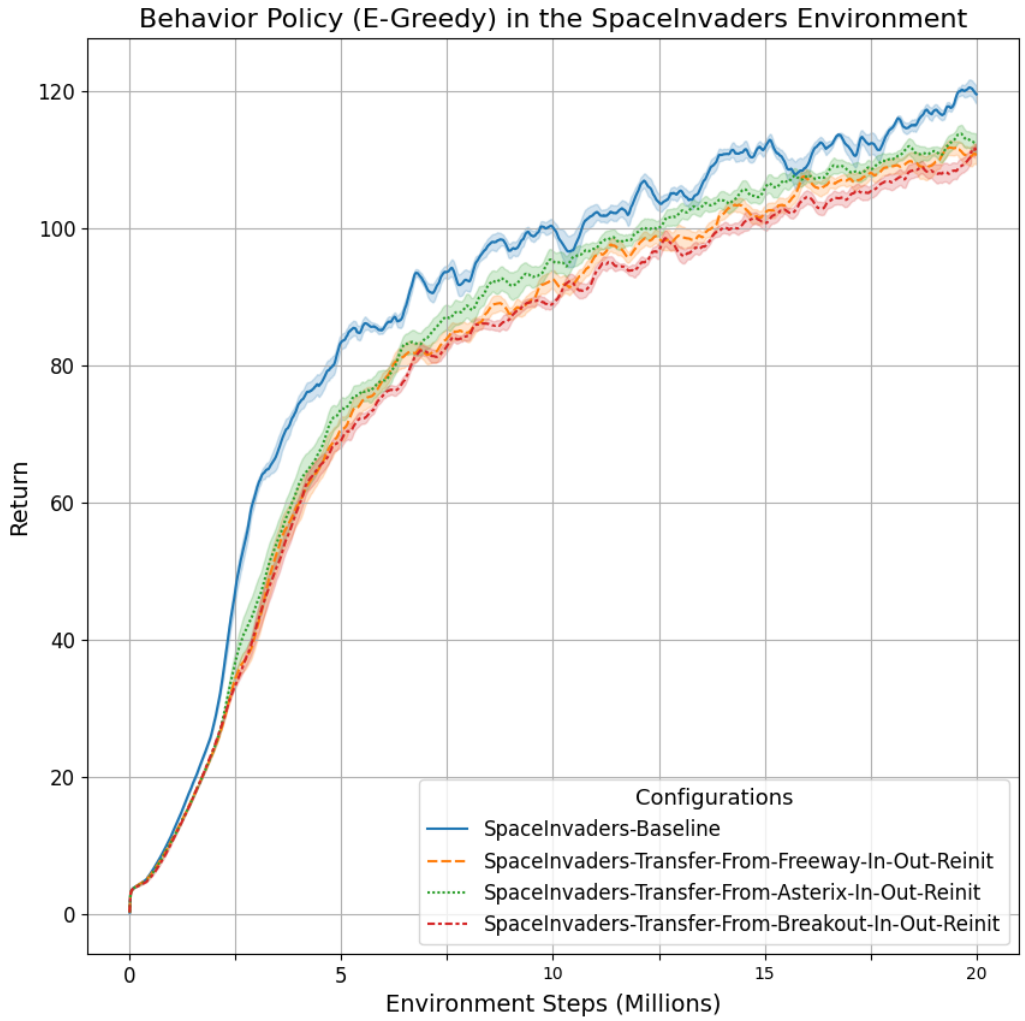

Figure B.5: Behavior Policy learning curves in the Space Invaders environment for various transfer configurations. Shown here are the average returns over 20 million steps.

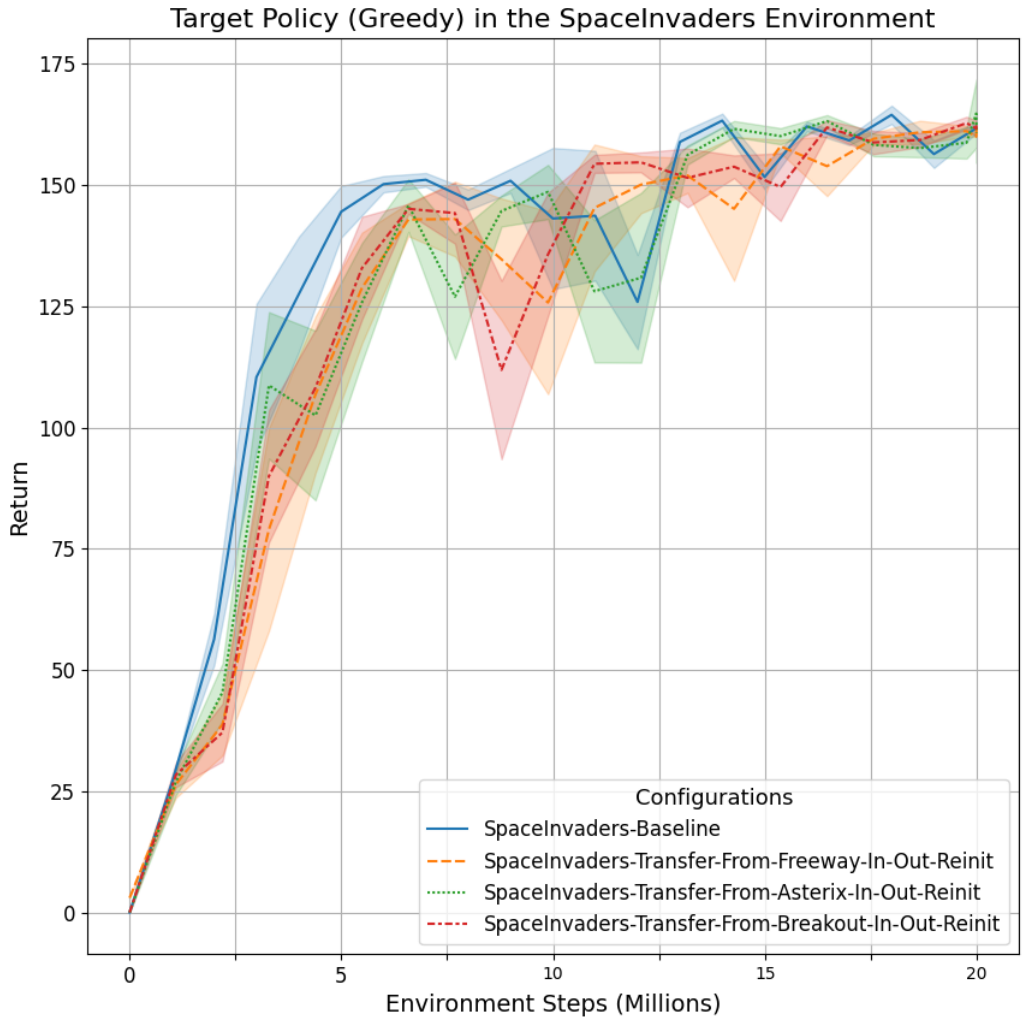

Figure B.6: Target Policy learning curves in the Space Invaders environment for various transfer configurations. The training was performed for 20 million steps, illustrating the evolution of mean returns.

ww