Exploring Transfer Learning in Deep Reinforcement Learning for Fighting Games

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

Bachelor of Computing Science (Honours)

By

Jason Vu
14265995

Supervisor: Mingshan Jia

University of Technology Sydney

March 2024

Abstract

Reinforcement learning (RL) agents have shown potential in learning complicated tasks, but they sometimes require prohibitively extensive training cycles, especially for applications involving a significant amount of data. Transfer learning is a strategy for reducing the time needed to train RL agents. This proposal describes an experiment to determine whether transfer learning can speed up a deep RL (DRL) variant training for learning and playing new games. The feature games used in this study are Street Fighter II and Fatal Fury 2, utilising the Gym Retro package inside a custom OpenAI Gym environment. The research approach involves two parts: first, training a proximal policy optimization (PPO) agent on Street Fighter II both with and without transferring pre-trained weights to evaluate the effect of transfer learning within the same game. Second, the approach compares training a PPO agent on Fatal Fury 2 with and without transferring pre-trained weights from Street Fighter II to assess generalisation capabilities. The premise is that transfer learning will shorten training time while preserving high performance across both contexts.

Contents

1 Introduction

Through trial-and-error interactions, intelligent agents can be trained to perform complex tasks using Reinforcement Learning (RL) (Sutton & Barto, 2022). However, learning effective policies from RL requires a lot of training and environment samples, which makes the process time-consuming. It may not always be able to gather enough information to start from scratch when training is necessary for an RL agent. Transfer learning seems to be one promising way to reduce the sample complexity of RL, in which it applies knowledge across tasks to improve learning on another different task. This experiment in this project proposal investigates exploring transfer learning to accelerate the training of deep reinforcement learning agents that are required to play games such as Street Fighter, which require highly technical players.The research question of the project proposal is: "Can transfer learning be used as a generalisable framework to improve the training time, performance and convergence rate of DRL agents?"

First, this paper will provide some background regarding transfer learning, proximal policy optimization (PPO), and reinforcement learning research performed in gaming environments. The methodology will then present a description of the proposal for training PPO agents with and without transfer learning for two scenarios: within the same game and between different games. Later, the importance and originality of this work are discussed. Finally, the conclusion summarises the expected contributions and future directions of sample-efficient reinforcement learning via transfer learning.

1.1 Research Aims and Objectives

The aim of this study is to determine whether agents using the proximal policy optimization (PPO) deep reinforcement learning algorithm can reach strong performance on a new game as well as the same game by moving learned weights from one game to another in less time. Training time to convergence and final agent performance on these games are the important metrics. The outcomes will show how transfer learning can be used to enable more sample-efficient reinforcement learning for gaming tasks.

This has significant implications for extending the application of RL to challenging real-world issues where sample efficiency is essential. Transfer learning has the potential to enable deep reinforcement learning for a broader range of applications by reducing the lengthy training time requirements.

The research objectives are:

- To see if using learned weights from one game can cut down the training time for a PPO agent to perform well on both that game and a new one, with a focus on Street Fighter II and Fatal Fury 2 as the test games.
- To figure out the main factors that affect how well transfer learning works for reinforcement learning (RL) in games, looking at things like how similar the source and target games are, how much pre-training data is available, and which transfer learning algorithm is used, specifically for fighting games.
- To create a useful and efficient transfer learning setup for reinforcement learning in gaming, especially for Street Fighter and similar fighting games, to make it easier to transfer knowledge and speed up the training of RL agents in different gaming environments.

By focusing on Street Fighter II and Fatal Fury 2 as target environments, this research aims to provide insights into the applicability and optimization of transfer learning techniques for RL in the domain of fighting games, ultimately contributing to the development of more efficient and adaptable gaming AI systems.

The expected procedure objectives are:

- Train a deep PPO agent using reinforcement learning on Street Fighter II as the source game. This agent will be our baseline with no transfer learning and will be trained until it reaches a strong, stable policy.
- Fine-tune a separate PPO agent on Street Fighter II, transferring the learned weights from the initial PPO agent to assess the transfer learning effect within the same game.
- Train a baseline PPO agent on Fatal Fury as the target game, continuing until convergence to a robust policy.
- Transfer the learned weights from the fine-tuned agent trained on Street Fighter II to a new PPOagent initialised with these weights for Fatal Fury 2.
- Evaluate and compare the training time and final performance of the transfer learning agents with the baseline agents trained only on Street Fighter II and Fatal Fury 2. If the transfer learning agents converge faster and/or perform better in the end, it will show that transfer learning helps speed up RL training for fighting games
- Investigate how key factors affect transfer learning effectiveness, such as the similarity between the source and target games, the amount of pre-training data, and the choice of transfer learning algorithm. I'll analyse how these factors impact the transferability of learned knowledge and the performance of transfer learning agents in fighting games.

1.2 Project Intention

The goal of this study is to gain a deeper comprehension of how transfer learning can be used to expedite the training of reinforcement learning agents for use in games. Researchers and practitioners working on RL applications for games and other domains will find the research's practical and efficient transfer learning framework for RL in gaming useful.

2 Background

Reinforcement learning (RL) is a method where an agent improves its performance through trial-and-error by interacting with an environment, aiming to maximise a numerical feedback signal. Instead of being instructed on which actions to take, the agent learns to identify the actions that provide the highest rewards through repeated experimentation (Sutton & Barto, 2022). Essentially, it is learning by trial and error. However, RL can be slow because it needs extensive interaction with the environment to gain enough experience (Zhu et al., 2023). Transfer learning (TL) helps address this issue by allowing the agent to use external knowledge to improve its learning process without needing as much interaction with the environment (Zhu et al., 2023). TL boosts RL's efficiency by sharing knowledge across different domains (Moos et al., 2022).

A notable example of applying reinforcement learning (RL) in gaming is demonstrated by training an AI team named OpenAI Five to excel in Dota 2 using the proximal policy optimization algorithm (OpenAI et al., 2019). Training RL agents for Dota 2 presents significant hurdles due to its prolonged training periods, limited visibility of the game state, and its high-dimensional nature. To address these challenges, researchers scaled up existing RL frameworks to unprecedented levels, employing thousands of GPUs over several months. They adopted traditional RL methods on an extensive scale, processing approximately 2 million frames every 2 seconds. Additionally, they developed distributed training systems and implemented continuous learning mechanisms to accommodate ten months of training. This achievement illustrates the capability of self-play RL to achieve superhuman performance in tackling complex tasks such as Dota 2.

A research gap lies in the extensive time needed to train agents for mechanically demanding games like Dota 2. Despite the success of OpenAI Five, its ten-month training regimen is resource-intensive. Transfer learning offers a promising solution by expediting training through the transfer of knowledge from one game to another. Leveraging an agent's experience across multiple games could yield time and cost savings while enhancing the efficiency and effectiveness of reinforcement learning (RL). Consequently, transfer learning emerges as a proposed approach to facilitate more streamlined knowledge transfer among RL gaming agents and to establish a comprehensive framework.

2.1 Overview of Deep Reinforcement Learning

2.1.1 Reinforcement Learning

The RL process can be explained using the framework of a Markov Decision Process (MDP) (Bellman, 1957). an agent interacts with an environment in discrete time steps. At each time step t, the agent observes the current state of the environment S_t, takes an action A_t, receives a numerical reward R_{t+1}, and transitions to a new state S_{t+1}, according to the dynamics of the environment.

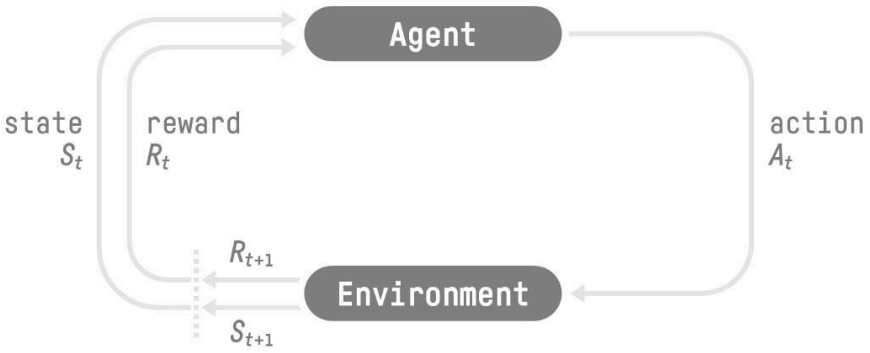

Figure 1. Reinforcement Learning Cycle (Sutton & Barto, 2022)

The RL properties, as formulated by the MDP, consists of the 5-tuple:

1. State (S): The set of all possible situations the agent can encounter. These states encapsulate all relevant information about the environment at a given time.
2. Action (A): The set of all possible moves or decisions the agent can take while in a certain state.
3. Transition (P): The transition function T(s, a, s') defines the probability of transitioning from state s to state s' after taking action a.
4. Reward (R): A reward function R(s, a, s') gives a reward to an agent for every transition from s to s′ via action a, which can be positive (reward) or negative (penalty)
5. Policy (π): A policy π gives an action for each state is the action recommended by the policy π for the agent to take when it is in state s.

Understanding the action space is crucial in our methodology, especially when using fighting games as examples. The action space refers to the complete set of possible actions within an environment and can be categorised as either discrete or continuous. In a discrete space, the number of potential actions is limited, while in a continuous space, the possibilities are infinite.

In games like Street Fighter and similar fighting games, the action space consists of a finite set of actions, including movements like left, right, up (jumping), and down (crouching), as well as basic attacks like punch and kick. Although there are more complex moves such as super moves and variations of punches and kicks, grasping the fundamental moveset is essential for understanding our transfer learning approach. It's important to note that Street Fighter and similar fighting games are episodic tasks, meaning they have a distinct starting point and ending point (a terminal state), typically indicated by factors like the timer running out or the health bar reaching zero. Each instance of gameplay forms an episode, comprising a sequence of states, actions, rewards, and subsequent states. Understanding this episodic nature provides insight into how our approach functions within the gaming environment.

The reward is fundamental in RL because it's the only feedback for the agent. Thanks to it, our agent knows if the action taken was good or not.

The cumulative reward at each time step t can be written as:

$$R(\tau) = r_{t+1} + r_{t+2} + r_{t+3} + r_{t+4} + \ldots + r_{t+n}$$

Where R(τ) is the return of cumulative reward, τ (Tau) is the trajectory—sequence of states and action.

This can also be written as:

$$R(\tau) = \sum_{k=0}^{\infty} r_t + k + 1$$

However, directly adding rewards may not be ideal, as early rewards are often more predictable and likely to occur than long-term ones. To address this, we introduce a discount rate known as gamma (ɣ), which ranges between 0 and 1, typically falling between 0.95 and 0.99. A higher gamma value implies a smaller discount, indicating that the agent prioritises long-term rewards. Conversely, a lower gamma value leads to a larger discount, suggesting a focus on short-term rewards.

Each reward is then discounted by gamma raised to the power of the time step. As the time step increases, the potential impact of future rewards diminishes, reflecting the agent's decreasing likelihood of experiencing those rewards.

The discounted expected cumulative reward can be written as:

$$R(\tau) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \ldots + \gamma^{n-1} r_{t+n}$$

Where gamma γ is the discount rate.

This can also be written as:

$$R(\tau) = \sum_{k=0}^{\infty} \gamma^k r_t + k + 1$$

2.1.2 Deep Reinforcement Learning

Deep reinforcement learning (DRL) merges reinforcement learning with deep neural networks, enabling the handling of high-dimensional state and action spaces (Li, 2018). In DRL, deep neural networks represent both the state space and policy, allowing for generalisation across similar states and actions. Techniques such as Deep Q-Networks (DQN) combine Q-learning with deep neural networks to approximate the optimal action-value function, while other algorithms that are also DRL like policy gradient, actor-critic algorithms and Proximal Policy Optimization (PPO) directly optimise policies to maximise expected cumulative rewards.

2.2 Deep Q-Learning

Deep Q-Networks (DQN) or Deep Q-Learning is a groundbreaking contribution in RL, introduced by (Mnih et al., 2015), to enable RL to work with deep neural networks. This is because with a nonlinear function like neural networks, it is known that RL is unstable or even divergent when the action value function is approximated (Li, 2018). DQN uses relay experience to stabilise the training of action value approximation using deep neural networks. The performance of DQN, after hyper parameter-tuning, outperformed previous RL algorithms such as linear function approximator, by 300% on Breakout and 1000% on Enduro, when testing and comparing using Atari games (Mnih et al., 2015). Because of this performance, using DQN will be considered for my RL agent, however PPO has much better compatibility as discussed later. The DQN method combines Q-learning with deep

neural networks. Q-learning is a model-free RL algorithm that learns the quality of actions in a given state. In DQN, a deep neural network (the Q-network) is trained to approximate the optimal action-value function (Q-function). The Q-learning update rule is derived from the Bellman equation, specifically the Bellman optimality equation for the action-value function. This can be mathematically represented as:

$$Q(s, a) \leftarrow Q(s, a) \ + \ \alpha[r \ + \ \gamma \ \max_{a'} \ Q(s', a') \ - \ Q(s, a)]$$

Where Q(s, a) represents the estimated action-value function, representing the expected future rewards for taking action a in state s, r is the immediate reward, ɣ is the discount rate, s′ is the next state, a′ is the next action, and α is the learning rate. The $\max_{a'} \ Q(s', a')$ is the maximum Q-value for the next state s′ over all possible actions a′. The part inside the parentheses, $r + \gamma \max_{a'} \ Q(s', a') \ - \ Q(s, a)$ represents the TD error, which measures the difference between the predicted Q-value and the target Q-value and helps in adjusting the Q-values towards more accurate estimations.

While DQN has shown significant advancements in reinforcement learning, its application is most effective in environments with discrete and low-dimensional action spaces. However, as discussed later in the paper, the multibinary action space of games like Street Fighter and Fatal Fury poses challenges that limit the effectiveness of DQN. This necessitated exploring alternative algorithms, such as Proximal Policy Optimization (PPO), which better align with the unique requirements of these environments.

2.3 Why DQN Won't Work for Game AI

Although DQN has proven effective for discrete action spaces in classic Atari games, its design does not align well with the multibinary action spaces of games like Street Fighter II and Fatal Fury 2. In these games, actions involve combinations of simultaneous button presses, such as holding "forward" and pressing "kick" together. Representing such combinations as single discrete actions leads to several issues:

- Exploding Action Space: The number of possible actions grows exponentially as the number of buttons increases. DQN would need to predict a Q-value for each possible combination, which becomes computationally infeasible in large action spaces.
- No Support for Multi-Binary Actions: DQN assumes actions are discrete and independent. In multibinary settings, where actions are represented as multiple simultaneous binary choices

(e.g., pressing or not pressing each button), DQN lacks the flexibility to model these interactions effectively.

These limitations, outlined in the Stable-Baselines3 documentation, make DQN unsuitable for environments requiring complex, high-dimensional control schemes.

2.4 Proximal Policy Optimization

To address the limitations of DQN, Proximal Policy Optimization (PPO) was chosen for this study. PPO, a policy-gradient-based method, is well-suited for environments with complex and multidimensional action spaces, such as the multibinary controls found in Street Fighter II and Fatal Fury 2.

PPO improves on traditional policy-gradient methods by addressing issues such as instability during updates and overly large policy shifts. It does so by introducing a clipped surrogate objective function, ensuring stable and efficient policy updates. This design makes PPO one of the most robust and widely used algorithms for reinforcement learning tasks (Schulman et al., 2017).

PPO is an on-policy method that directly optimises the policy $\pi_\theta(a|s)$, which represents the probability of taking an action a given state s. The goal of PPO is to maximise the expected cumulative reward by updating the policy parameters θ, while ensuring updates remain stable and bounded.

The PPO objective function is derived from the trust-region policy optimization (TRPO) algorithm but simplifies its constraints. The PPO objective function is:

$$L^{CLIP} = \hat{E}_t\left[min\left(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t\right)\right]$$

Where:

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio between the new and old policies.
- $\hat{A}_t$ is the advantage function, which estimates how much better or worse an action is compared to the average action taken at state s_t.
- ϵ is a hyperparameter that controls the clipping range, ensuring that updates to $r_t(\theta)$ do not exceed a certain threshold.

The clip() function ensures stability by constraining the policy update to a trust region. If the probability ratio $r_t(\theta)$ deviates too far from 1 (indicating a large policy change), the objective is clipped, effectively preventing overly aggressive updates (Schulman et al., 2017).

PPO can handle environments where actions consist of multiple binary decisions, such as pressing or not pressing individual buttons. The multibinary action space of fighting games involves combinations of button presses that DQN cannot handle effectively. PPO's support for continuous and multidimensional action spaces aligns well with the needs of this study. The clipped objective ensures training stability, even in dynamic environments where small changes in policy can have significant impacts on performance. Furthermore, PPO has demonstrated success in numerous RL tasks, from robotic control to game-playing environments, making it a reliable choice for challenging domains (Yu et al., 2022).

2.5 Transfer Learning

Transfer Learning in deep learning involves using a pre-trained model, which has been trained on a large dataset, to solve a different but related problem. This technique allows the transfer of learned features and knowledge from one task (source domain) to another (target domain), significantly reducing the need for extensive labelled data and training costs (Zhuang et al., 2020). The formal definition of Transfer Learning can be defined as:

Definition 1. ***(Transfer Learning)***: Given a learning task T_t based on D_t where T = {y, $f(x)$ }, y is the label function, $f(x)$ is the predictive function and T_t is the target task; D = {$\mathcal{X}$, P(X)}, $\mathcal{X}$ is the feature space, P(X) is the edge probability distribution and D_t is the target domain. Transfer learning aims to improve the performance of predictive function f(x) for the learning task T_t by transferring the weights from the source domain D_s and and source task T_s. However, for transfer learning on deep neural networks, the f(x) must be a nonlinear function.

In practice, transfer learning is typically implemented by using the early layers of a pre-trained network as a fixed feature extractor. Only the later layers of the network are retrained on the new task, allowing the model to adapt to specific needs without starting from scratch. This is particularly beneficial when the new task has limited data available, as it utilises the generalised representations learned from the large source dataset (Tan et al., 2018). This process can be described as:

$$T_t = Train[D_t, Update(T_s, D_s)]$$

Where Update(T_s, D_s) represents the process of reusing the source model and fine-tuning it to suit characteristics of the target domain.

3. Related Work

3.1 Reinforcement Learning using Transfer Learning

Applications of transfer learning (TL) in reinforcement learning (RL) include knowledge transfer across source and target tasks, domains, and agents (Zhu et al., 2023). By using and transferring information from previously taught RL agents or external knowledge domains, TL approaches in RL aim to improve the speed, efficiency, and efficacy of training new agents.

In the paper "Improving DQN Training Routines with Transfer Learning", He et al. (2020) present an application of TL to deep reinforcement learning and two Atari games, Breakout and Pong. The aim of the study is to explore whether it is possible to improve the learning outcomes by applying a pre-trained DQN from one source gaming environment to another target environment. To check this, they first reproduce the convolutional layers of a DQN agent learning in a new target game environment with the stored weights of convolutional layers from a DQN agent that was trained from scratch in that game environment. The authors make use of OpenAI Gym-a suite of tools for building and contrasting RL algorithms-to produce the game scenarios and testbed. With OpenAI Gym, it is easy to replace games with stored network modules according to the modular architecture of the authors and to reset weights as appropriate. They utilise double duelling DQN with experience replay and set goals in terms of DQN model design.

The primary discovery of their research is that the DQN agent's initial learning performance in the target task is significantly improved by transferring knowledge from the low-level convolutional layers. On the other hand, the results for the IMPALA agent are still unclear. To put it another way, the convergence acceleration obtained by moving weights from the first convolutional layer of a DQN agent trained on Breakout to another DQN agent learning Pong was around double that of training without transfer learning. However, over time, the maximum mean rewards produced by the traditional training methods and the transfer learning methodology were comparable.

3.2 Other Transfer Learning Technique

Another transfer learning method for reinforcement learning that makes use of adversarial objectives and data augmentation is proposed in the paper "Deep Reinforcement Learning with Adversarial

Objective and Augmentation" (Hsu et al., 2018). In order to prevent a domain discriminator from discriminating between encoded target observations and source instances, their solution involves learning a mapping function that encodes target task observations into the source task feature space. They also found that knowledge transfer is facilitated by efficient data augmentation during source task training. Their technique accelerates learning on a new target task by leveraging prior knowledge from a task, and it is incorporated into a DQN architecture. Unlike previous efforts that required several trained agents on various source tasks, this method can accelerate target task training with only one prior job. Learning is further accelerated by simple data augmentation. This answers the research question by showing how transfer learning can shorten the time needed for RL training, effectively increasing the possibility of it being a generalised framework. The method's exclusive evaluation of Atari games, with no link to actual examples, is a drawback. This method could be used to validate and adapt to more complicated real-world reinforcement learning situations in future study. All things considered, by recycling data from a single source task, the study presents a unique adversarial transfer learning technique for accelerating deep reinforcement learning training.

4 Future Impact

4.1 Significance

This research is important because it addresses a major obstacle in reinforcement learning (RL), namely the long training time needed for RL agents to acquire complicated tasks. Because this research creates a useful and efficient framework for transfer learning for reinforcement learning in gaming, it has the potential to accelerate the development of reinforcement learning applications in several domains such as robots, finance, and healthcare. Deep RL has great potential for training intelligent agents, as was previously mentioned, but its practical use is limited by the large sample complexity. This study will show that transfer learning may reduce training times, which will instantly boost the viability of using RL to complex tasks like gaming. Additionally, Deep RL can be used to construct proficient agents in additional domains with faster and more sample-efficient training. Thanks to this discovery, the gaming industry may now employ sophisticated reinforcement learning algorithms without having to pay unaffordably high processing costs. The necessity for months of training with every new game is eliminated by the ability to transfer learnt information between titles. This will enable developers of video games to take use of reinforcement learning's advantages in tasks like autonomous game testing, adaptive game AI, and generative content creation.

4.2 Contribution to Technological Advancement

This will be achieved through a novel and unique approach to transfer learning in reinforcement learning, which has great potential for enhancing technical proficiency and practise both within academia and society. The proposed system will be tested on two combat games due to the demanding and well-defined benchmark they provide for RL algorithm testing. Conferences and scholarly journals will be used to present the findings of this study. Furthermore, the developed framework will be publicly available to researchers and practitioners who want to use it. Technically, this work will provide new insights about which aspects of deep reinforcement learning knowledge transfer most effectively between game scenarios. Rules of efficient information transfer in reinforcement learning will be found by studying the influences of different source and target games.

4.3 Contribution to Society

The proposed study might benefit industry and society in several ways. Firstly, by accelerating the development of RL applications, this study may create new and innovative goods and services that would enhance peoples' lives. Examples of applications could also include the use of RL in developing new medical equipment or techniques to treat and diagnose illnesses, self-governing machines or robots able to pick up complex tasks in dangerous real-world settings, and new algorithms for trading or risk management systems for the financial industry.

4.4 Reimagining the Future

The suggested study will help reimagine the future in a number of ways. First, it offers a brand-new RL transfer learning model that is especially designed for game-related tasks. The foundation of this framework is the realisation that many games within the same genre share elements in common, such as in fighting games, there is 4-way movement, kick, punch and block. As shown in "Improving DQN Training Routines with Transfer Learning" (He et al., 2020) showing a double in acceleration training time, the suggested framework can dramatically shorten the amount of time needed for RL agents to learn new games by sharing knowledge across fighting games. Of course, fighting games are niche but the whole idea of this research is to develop a framework and prove that using Transfer Learning alongside with Reinforcement Learning can be generalised, and potentially used outside of the gaming sphere.

Second, as mentioned in "Deep Reinforcement Learning with Adversarial Objective and Augmentation" (Hsu et al., 2018), the proposed work investigates the application of adversarial training to improve the transferability of RL agents. It has been discovered that using adversarial training is a helpful tactic for making machine learning models more resilient. The agent acquires more

comprehensive feature representations that are unaffected by minute changes in the input data by being trained to be resistant to these hostile circumstances. Because its rules rely less on surface patterns, the resultant agent can adapt to various contexts and jobs more easily.

Finally, the proposed research will contribute significantly to the field of RL by improving understanding of the elements that determine transfer learning effectiveness. The findings of this study will be utilised to provide guidelines and best practices for implementing transfer learning in RL applications. For example, the findings of "Improving DQN Training Routines with Transfer Learning" (He et al., 2020) show that transferring only early convolutional layers can increase initial learning while offering guidance on selective transfer. In summary, the proposed methodologies, theories, and findings reflect novel approaches to improving deep reinforcement learning via strategic transfer learning, adversarial training, and an examination of the important elements impacting transfer efficacy. This will eventually progress the state-of-the-art in using transfer learning on deep RL agents.

5 Methodology

Initially, DQN was considered as the RL algorithm for this study due to its established success in discrete action environments. However, during preliminary analysis, it became evident that the multibinary action space of the target environments, coupled with the need for simultaneous button presses, posed challenges that DQN could not address effectively. As a result, PPO was selected for its compatibility with multibinary actions, stability, and flexibility.

This research will employ a laboratory experiment designed to evaluate the effectiveness of transfer learning in accelerating the training of reinforcement learning (RL) agents in fighting games. For this purpose, AI agents were trained to accomplish a specific objective: defeating a designated opponent in a single-stage matchup. In Street Fighter II, the agent-controlled character, Ryu, faced off against Guile in the same stage for each episode. Similarly, in Fatal Fury 2, the AI-controlled character, Terry, competed against Andy under identical conditions. By limiting the training scope to single-match scenarios rather than full-game playthroughs, the experiments ensured a focused and consistent evaluation environment. The experiment will involve two parts: first, training a PPO agent on Street Fighter II both with and without transfer learning to assess the impact within the same game. Second, the study aims to test the generalisation of transfer learning and will involve training two PPO agents on a different fighting game, Fatal Fury 2, one with a baseline training and the other with transferred weights from Street Fighter II, comparing one agent trained with transfer learning and another

without. The final performance of the agents is measured in win rate and mean reward will be compared to determine the benefits of transfer learning in both contexts. This setup ensures that the evaluation is pertinent to the mechanics and challenges specific to fighting games.

5.1 Methodology Flowchart

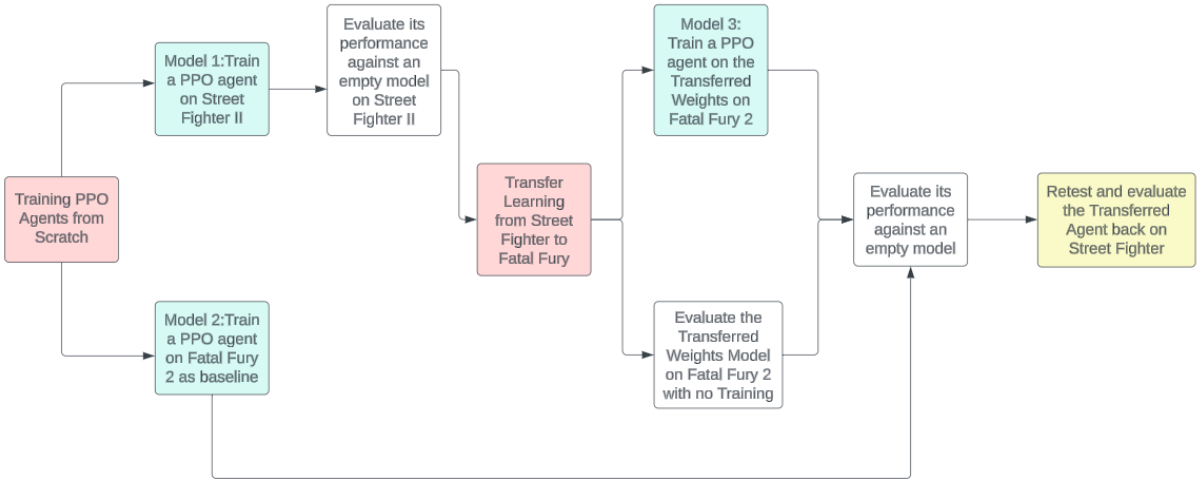

Figure 2. Methodology Flowchart

5.2 Part 1: Training the Baseline Model on Street Fighter II

Game Selection:

The source and target game for this part is Street Fighter II. This game was chosen due to its distinct mechanics and multi-binary action space, making it an excellent choice for evaluating reinforcement learning (RL) techniques.

Setting Up Gym Retro Environment:

The Gym Retro environment was used to emulate Street Fighter II, providing a standardised framework for RL agent training. The environment was set up using NoMachine to access a virtual Linux-based server equipped with an NVIDIA L4 GPU. A Python 3.9 virtual environment was created, and all necessary dependencies were installed, including gymnasium (a maintained fork of OpenAI Gym), stable-baselines3, and other essential libraries listed in a requirements.txt file such as Pytorch version 12.1. Gymnasium and retro were imported first to allow access to ensure compatibility and reproducibility for RL training. Both Street Fighter II and Fatal Fury 2 ROMs were installed from

the web and loaded into the virtual environment by placing them in a ROMS folder and running the command:

```
python -m retro.import .
```

Exploratory Data Analysis

To understand the compatibility and similarity between Street Fighter II and Fatal Fury 2, an exploratory data analysis was conducted by loading in both games and creating a simple one line environment. This analysis focused on the games' control schemes, observation space, and action space.

- Button Mapping: Both games share the same button mappings ['B', 'A', 'MODE', 'START', 'UP', 'DOWN', 'LEFT', 'RIGHT', 'C', 'Y', 'X', 'Z']:

```
env.buttons
✓ 0.0s
['B', 'A', 'MODE', 'START', 'UP', 'DOWN', 'LEFT', 'RIGHT', 'C', 'Y', 'X', 'Z']
```

- Action Space: Both games utilise a MultiBinary(12) action space, meaning there are 12 discrete binary actions representing button presses and combinations. This means that there are 12 positions, varying permutations of 0 and 1 for valid actions, that particular action may correspond to particular move the agent can make in the game e.g. [1,0,1,0,1,0,1,0,1,0,1,0] may correspond to a punch move. Street Fighter II and Fatal Fury 2 have the same action space of 12 positions. Every possible move the agent can make in the game is represented by a binary number, therefore it ignores motion control inputs/sequence and just randomises the action space for each step e.g. instead of the AI agent having to press down, down-forward, forward, punch to perform a Hadouken, it can just press a single button to perform the move (if the binary number matches the Hadouken move)

```
env.action_space
✓ 0.0s
MultiBinary(12)
```

```
# Sample the actions that are avaialble - MultiBinary
env.action_space.sample()
✓ 0.0s
array([1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0], dtype=int8)
```

- Action Combinations: Valid action combinations for both games are:

```
env.button_combos
✓ 0.0s
[[0, 16, 32],
 [0, 64, 128],
 [0, 1, 2, 3, 256, 257, 512, 513, 1024, 1026, 1536, 2048, 2304, 2560]]
```

- Observation Space The observation space for both games is represented as a Box, allowing for the resizing and preprocessing of frames, making the input suitable for RL training.

```
# Sample the observation space
env.observation_space
✓ 0.0s
Box(0, 255, (200, 256, 3), uint8)
```

Custom Environment for Street Fighter II:

A custom environment was developed to tailor the agent's training to the nuances of the game.

```
class StreetFighter(Env):
    def __init__(self):
        super().__init__()
        self.game = retro.make(game='StreetFighterIISpecialChampionEdition-Genesis', render_mode=None, use_restricted_actions=retro.Actions.FILTERED)
        self.observation_space = Box(low=0, high=255, shape=(84, 84, 1), dtype=np.uint8)
        self.action_space = MultiBinary(12)
```

Some environment tweaks included:

- Filtered Action Space: The action space was restricted to valid button combinations that resulted in normal or special moves. This reduced the agent's exploration of irrelevant or invalid actions, streamlining learning and improving efficiency.
- Rendering Disabled: Rendering was set to None during training to increase the frames per second (FPS), significantly speeding up the training process by bypassing the computational cost of visual output.

The Step Function which includes the Frame Delta, Reward Function and termination:

```
def step(self, action):
    obs, reward, terminated, truncated, info = self.game.step(action)
    obs = self.preprocess(obs)

    # Frame delta
    frame_delta = obs - self.previous_frame
    self.previous_frame = obs

    # Custom hp-based reward to encourage defense and attack
    if (info['health'] == 176 and info['enemy_health'] == 176) or (info['health'] == 0 and info['enemy_health'] == 0) or info['health'] == 0:
        reward = 0
    else:
        # Reward is the hp changes
        reward = (self.previous_enemy_health - info['enemy_health'])*2 - (self.previous_health - info['health'])

    self.previous_enemy_health = info['enemy_health']
    self.previous_health = info['health']

    return frame_delta, reward, terminated, truncated, info
```

- Frame Delta: The input to the agent was the difference between consecutive frames, providing information on changes in the environment, which improved the agent's performance.
- Reward Function: A bespoke reward function was implemented:
    - Positive Rewards: Proportional to the reduction in the opponent's health.
    - Negative Rewards: Proportional to the reduction in the agent's health, acting as a penalty.
- Done/Termination: Episodes ended when either the agent or the opponent won 2 rounds, ensuring the agent was trained on the dynamics of single-match outcomes. This required modifying Gym Retro's scenario file on Street Fighter II.

The render function for rendering the game:

```
def render(self):
    self.game.render()
```

The reset function for resetting the game when an episode ends:

```
def reset(self, *, seed=None):
    # Frame delta
    obs, info = self.game.reset()
    obs = self.preprocess(obs)
    self.previous_frame = obs

    # Create initial variables
    self.previous_enemy_health = 176
    self.previous_health = 176

    return obs, info
```

The preprocess function for preprocessing the game, further details is explained in the next section:

```
def preprocess(self, observation):
    gray = cv2.cvtColor(observation, cv2.COLOR_BGR2GRAY)
    resize = cv2.resize(gray, (84,84), interpolation=cv2.INTER_CUBIC)
    state = np.reshape(resize, (84,84,1))
    return state
```

The close function for closing the game when everything is done:

```
def close(self):
    self.game.close()
```

Data Preprocessing:

The observation space was formatted as a Box, allowing frames to be resized during preprocessing. This ensured compatibility with grayscale conversion and reduced computational overhead. Game frames were preprocessed primarily using grayscale conversion–requiring to import OpenCV, which reduced the computational overhead while retaining critical visual details and frame resizing which reduced the dimensions. No frame skipping was applied in this setup. The raw game frames, originally in RGB format with dimensions 200×256×3, were preprocessed to a smaller size of 84×84×1, where height and width are reduced to 84x84 pixels to decrease dimensionality and computational complexity. The channel is set to 1, as the image was converted to grayscale. Also, stablebaseline3 requires the image to be in the shape of (84, 84, 1) (height, width, channel). These preprocessing techniques align with the methods used in Mnih et al. (2013), where similar steps were applied in training deep Q-networks on Atari games to reduce computational costs and enhance learning efficiency.

Figure 3. Original Observation Space of SF2

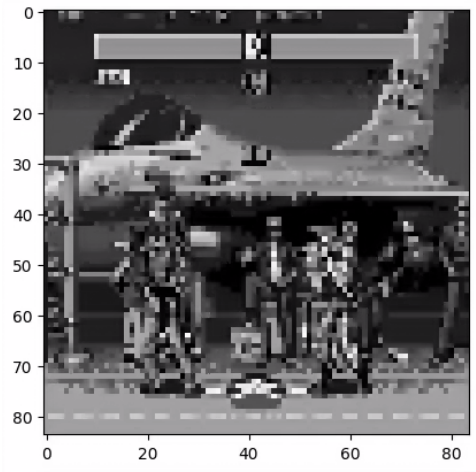

Figure 4. Preprocessed Observation Space of SF2

We can see that in Figure 4, the image of the first frame is grayscale and resized.

A preprocessing function was implemented as part of the custom environment to handle these transformations.

Running plt.imshow(cv2.cvtColor(obs, cv2.COLOR_BGR2RGB)) after creating an instance of the custom environment, we can manually see what the AI agent will see changes in pixels for each frame or timestep, which is what will be used for training. Frame 1 shows movement from both players, frame 2 has no movement so, no pixel change is recorded, and frame 3 and 4 has slight pixel change, indicating a movement in limbs.

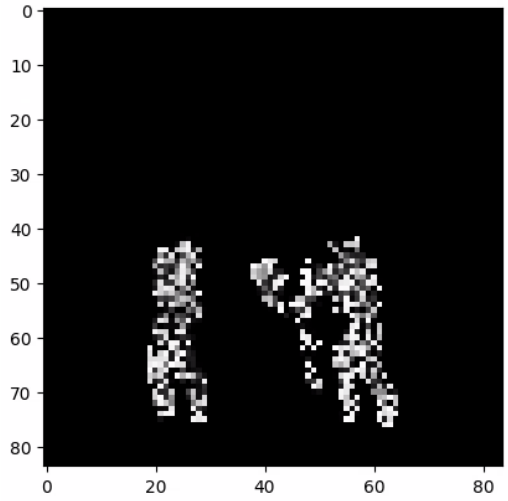

Figure 5. Frame 1

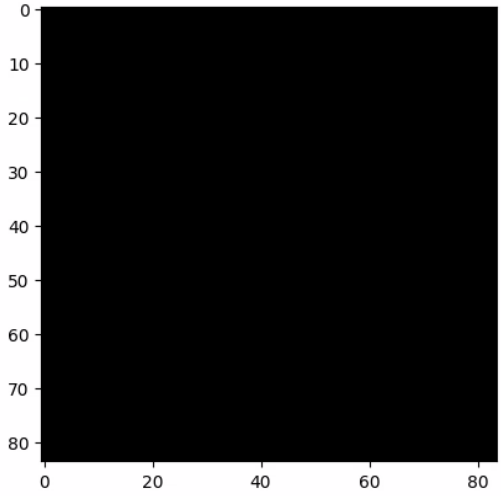

Figure 6. Frame 2

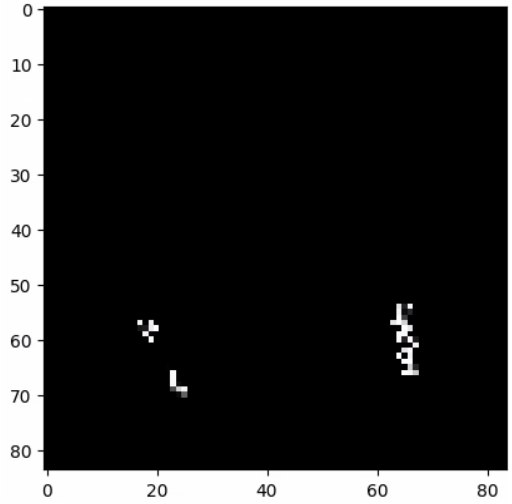

Figure 7. Frame 3

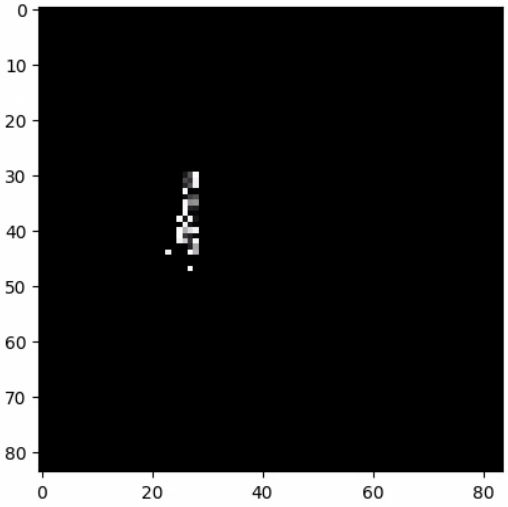

Figure 8. Frame 4

Hyperparameter Tuning with Optuna:

Optuna was used for hyperparameter optimization. Twenty trials were conducted, each running for 100,000 timesteps then using 20 game episodes to evaluate the average reward. The trials with the highest mean reward will be the model hyperparameter that will be used for the rest of the models for consistency. The optimization process efficiently searched the parameter space to determine the best configuration for the PPO algorithm. The hyperparameter ranges were first set up then the trial function afterwards. The hyperparameters for the PPO algorithms that were used included the n_steps, gamma, learning_rate, clip_range, gae_lambda and ent_coef.

```python
def optimize_ppo(trial):
    return {
        'n_steps': trial.suggest_int('n_steps', 2048, 8192),
        'gamma': trial.suggest_loguniform('gamma', 0.8, 0.9999),
        'learning_rate': trial.suggest_loguniform('learning_rate', 1e-5, 1e-4),
        'clip_range': trial.suggest_uniform('clip_range', 0.1, 0.4),
        'gae_lambda': trial.suggest_uniform('gae_lambda', 0.8, .99),
        'ent_coef': trial.suggest_uniform('ent_coef', 0.01, 0.02)
    }
```

```python
def optimize_agent(trial):
    try:
        model_params = optimize_ppo(trial)

        # Create environment
        env = StreetFighter()
        env = Monitor(env, LOG_DIR)
        env = DummyVecEnv([lambda: env])
        env = VecFrameStack(env, 4, channels_order='last')

        # Create algo
        model = PPO('CnnPolicy', env, tensorboard_log=LOG_DIR, verbose=0, **model_params, device='cuda')

        model.learn(total_timesteps=100000)

        # Evaluate model
        mean_reward, _ = evaluate_policy(model, env, n_eval_episodes=10, deterministic=True)
        env.close()

        SAVE_PATH = os.path.join(OPT_DIR, 'trial_{}_best_model'.format(trial.number))
        model.save(SAVE_PATH)

        return mean_reward

    except Exception as e:
        print(f"An exception occurred: {e}")
        traceback.print_exc()  # Full stack trace
        return -1000
```

```python
study = optuna.create_study(direction='maximize')
study.optimize(optimize_agent, n_trials=20, n_jobs=1)
```

Training Baseline PPO Agent:

The baseline PPO agent was trained using stable-baselines3. To ensure the agent could efficiently handle the complex dynamics of the game:

```
env = StreetFighter()
env = Monitor(env, LOG_DIR)
env = DummyVecEnv([lambda: env])
env = VecFrameStack(env, 4, channels_order='last')

model = PPO('CnnPolicy', env, tensorboard_log=LOG_DIR, verbose=1, **model_params)

model.learn(total_timesteps=5000000, callback=callback)
```

- Vectorised Environment: The environment was vectorised in a dummy vector which is required for frame-stacking.
- Frame Stacking: The agent was provided with a stack of the last four frames, giving it temporal context for decision-making. Frame stacking, as highlighted in Mnih et al. (2015), has been proven effective in capturing motion and temporal dependencies in RL tasks involving visual inputs reprocessing strategies were foundational in enabling the reinforcement learning agent to operate effectively within the constraints of computational resources while maintaining high performance. Especially for a game very dependent on frame data such as Street Fighter, being able to react to a move as the frames come out requires the AI to remember these frames thus the stacking of four previous frames in one timestep.
- The agent was trained for 5 million timesteps (or per in game frames) to allow it to develop a robust and effective policy.
- A custom setup callback function was set up to ensure we can retrieve a model from any timestep of the training where the performance is peaking or may need some parameters tweaking.

```
class TrainAndLoggingCallback(BaseCallback):

    def __init__(self, check_freq, save_path, verbose=1):
        super(TrainAndLoggingCallback, self).__init__(verbose)
        self.check_freq = check_freq
        self.save_path = save_path

    def _init_callback(self):
        if self.save_path is not None:
            os.makedirs(self.save_path, exist_ok=True)

    def _on_step(self):
        if self.n_calls % self.check_freq == 0:
            model_path = os.path.join(self.save_path, 'best_model_{}'.format(self.n_calls))
            self.model.save(model_path)

        return True
```

Evaluation:

The trained agent's performance was evaluated over 100 episodes by loading in the model and using a custom evaluation function using stablebaseline3 which includes the following metrics:

- Mean Reward: The average reward achieved per episode, reflecting the agent's ability to maximise positive outcomes during gameplay.
- Win Rate: The percentage of episodes in which the agent won 2 rounds against the CPU opponent. Each episode concluded when either the agent or the CPU won 2 rounds.

```python
def evaluate_with_stats(model, env, n_eval_episodes=100, deterministic=False):
    wins = 0
    total_rewards = []

    for episode in range(n_eval_episodes):
        obs = env.reset()  #
        done = False
        episode_reward = 0

        while not (done):
            # Predict action based on model
            action, _ = model.predict(obs, deterministic=deterministic)
            obs, reward, done, info = env.step(action)

            # Extract the relevant environment info
            if isinstance(info, list):  # For vectorized environments
                info = info[0]

            # Accumulate reward
            episode_reward += reward

            # Check for win condition in `info` at the end of the episode
            if done and info.get('matches_won', 0) == 2:
                wins += 1

        total_rewards.append(episode_reward)

    # Calculate statistics
    mean_reward = sum(total_rewards) / n_eval_episodes
    win_rate = wins / n_eval_episodes

    return mean_reward, win_rate
```

```python
model = PPO.load('./train_1/best_model_3240000.zip')

mean_reward, win_rate = evaluate_with_stats(model, env, n_eval_episodes=100)
```

Comparison with an Untrained Model:

To assess the impact of training, an untrained baseline model (initialised with the same model parameters but without any training) was tested under identical conditions. The performance of the untrained model was evaluated across the same metrics—mean reward and win rate—to provide a benchmark for comparison. The empty model was created with the same configuration as the trained models.

```
empty_model = PPO(
    'CnnPolicy',  # Use the same policy as the trained model
    env,          # Same environment
    tensorboard_log=LOG_DIR,
    verbose=1,
    **model_params  # Use the same hyperparameters
)

mean_reward_empty, win_rate_empty = evaluate_with_stats(empty_model, env, n_eval_episodes=100)
```

Results Analysis:

The evaluation included a detailed comparison of the trained and untrained models which was performed using matplotlib for data visualisation and graphs generated by Tensorboard while training the model:

- Performance Improvement: Differences in mean reward and win rate between the trained and untrained models were computed to quantify the effect of training.

5.3 Part 2: Testing Generalisation Capability to a Different Game

Game Selection:

The source game was Street Fighter II, using the pre-trained agent from Part 1, and the target game was Fatal Fury 2. These games were chosen due to their similar mechanics and action spaces, making them suitable for testing transfer learning while providing distinct challenges to assess generalisation capabilities.

Setting Up Gym Retro Environment:

The Gym Retro environment was configured to support Fatal Fury 2, using the same setup as in Part 1. The environment ran on a Linux-based server with an NVIDIA L4 GPU and Python 3.9 virtual environment, ensuring consistent experimental conditions.

Data Preprocessing and Custom Environment Creation:

To maintain consistency with Street Fighter II, a custom environment was created for Fatal Fury 2. This environment replicated the reward function from the Street Fighter II environment, where:

- Positive Rewards: Reward was based on the reduction in the opponent's health.
- Penalties: A penalty was applied for reductions in the agent's own health.

The only difference between the two environments was the variable names, as the Fatal Fury 2 game's information data format differed. Also, the health points (HP) in Fatal Fury 2 is different from Street Fighter II, having a HP of 96 while Street Fighter II has 176. The HP in Fatal Fury 2 has been

normalised to match Street Fighter II by multiplying the reward by (176/96). However, the core mechanics and health-based reward system remained identical.

```python
class FatalFury(Env):
    def __init__(self):
        super().__init__()
        self.game = retro.make(game='FatalFury2-Genesis', render_mode=None, use_restricted_actions=retro.Actions.FILTERED)
        self.observation_space = Box(low=0, high=255, shape=(84, 84, 1), dtype=np.uint8)
        self.action_space = MultiBinary(12)

    def step(self, action):
        obs, reward, terminated, truncated, info = self.game.step(action)
        obs = self.preprocess(obs)

        # Frame delta
        frame_delta = obs - self.previous_frame
        self.previous_frame = obs

        # Custom hp-based reward to encourage defense and attack
        if (info['health'] == 96 and info['opponent_health'] == 96) or (info['health'] == 0 and info['opponent_health'] == 0) or info['health'] == 0:
            reward = 0
        else:
            # Reward is the hp changes
            reward = (176/96)*((self.previous_enemy_health - info['opponent_health'])*2 - (self.previous_health - info['health']))


        self.previous_enemy_health = info['opponent_health']
        self.previous_health = info['health']


        return frame_delta, reward, terminated, truncated, info
```

Additionally, the following preprocessing techniques were still applied:

- Observation space resizing
- Grayscale Conversion: Frames were converted to grayscale to reduce complexity.
- Frame Deltas: The AI agent was provided with frame deltas (current frame minus the previous frame) for better performance.
- Frame Stacking: The last four frames were stacked to give the agent temporal context for decision-making.

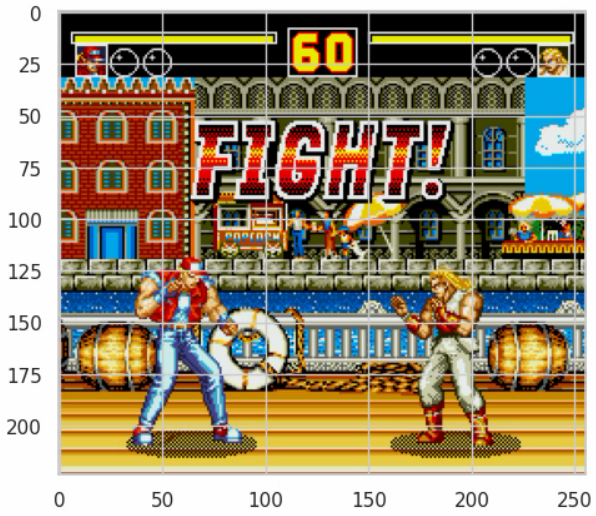

Figure 9. Original Observation Space of FF2

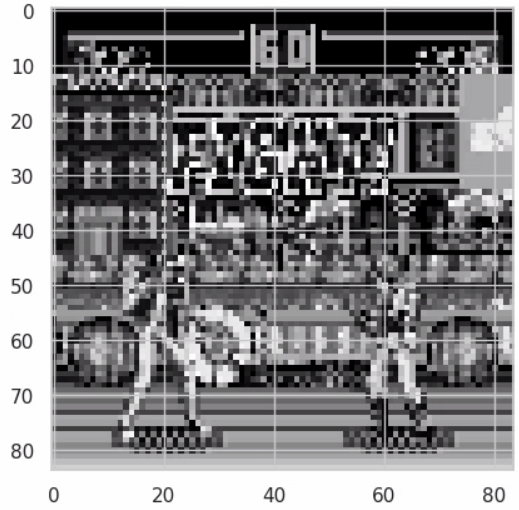

Figure 10. Preprocessed Observation Space of FF2

Training Models on Fatal Fury 2

1. Model Trained from Scratch:

   A baseline PPO agent was trained on Fatal Fury 2 from scratch using the hyperparameters optimised in Part 1 for Street Fighter II. Training was conducted for 5 million timesteps using stable-baselines3 to establish a robust policy. The code is the same as Street Fighter but uses the Fatal Fury custom environment instead.

2. Model Trained with Transferred Weights:

   Weights from the pre-trained Street Fighter II model were transferred to initialise a new PPO agent for Fatal Fury 2. This model was then fine-tuned on Fatal Fury 2 for 5 million timesteps, using the same environment and hyperparameters as the model trained from scratch. This step is the transfer learning part where this can be done by loading in the Street Fighter II trained model into the Fatal Fury custom environment. Then using that model to learn via stablebaseline3 reinforcement learning training.

```
env = FatalFury()
env = Monitor(env, LOG_DIR)
env = DummyVecEnv([lambda: env])
env = VecFrameStack(env, 4, channels_order='last')

model = PPO.load('./train_1/best_model_3240000.zip', env=env, tensorboard_log=LOG_DIR) # Street Fighter II Model

model.learn(total_timesteps=5000000, callback=callback)
```

3. Street Fighter II Model Without Training on Fatal Fury 2:

   The pre-trained Street Fighter II model was directly evaluated on Fatal Fury 2 without any additional training. This test assessed whether the agent's performance carried over to the new game without adaptation.

4. Untrained Model:

   An untrained model with random initial weights was also evaluated on Fatal Fury 2 to establish a baseline performance with no prior training. The code is the same as the empty model in the Street Fighter II part with the same model parameters as every other model.

Evaluating Generalisation and Transfer Learning

Evaluation Metrics:

Each model was evaluated over 100 episodes on Fatal Fury 2, using the following metrics:

- Mean Reward: Average reward per episode, reflecting the agent's ability to maximise positive outcomes during gameplay.
- Win Rate: Percentage of episodes where the agent won 2 rounds against the opponent, concluding the match.

Comparative Testing:

- Baseline Comparison: The performance of the model trained from scratch on Fatal Fury 2 was compared with the model trained using transferred weights to evaluate the effectiveness of transfer learning.
- Generalisation Evaluation: The performance of the pre-trained Street Fighter II model on Fatal Fury 2 was compared with both the untrained model and the trained agents to understand its cross-game generalisation capabilities.

Testing Retention of Learned Policies

To evaluate the impact of transfer learning on the retention of learned skills, the model trained on Fatal Fury 2 with transferred weights was reloaded into the Street Fighter II environment and evaluated over 100 episodes. This test assessed whether training on Fatal Fury 2 affected the performance of the agent on its original game.

Results Analysis

A statistical analysis was conducted to determine the effectiveness of transfer learning:

- Performance Comparisons: The mean reward and win rate of all models were compared to assess the advantages of training with transferred weights versus training from scratch. Episode mean length was also analysed to see changes in training behaviour.
- Generalisation Evaluation: The performance of the pre-trained Street Fighter II model on Fatal Fury 2 was compared with both the untrained model and the trained agents to understand its cross-game generalisation capabilities.

5.4 Adaptability of Methodology

The methodology for this study was designed to be flexible, allowing adjustments to address unforeseen challenges or outcomes effectively. This adaptability ensures that the research process remains robust and efficient, even when unexpected issues arise.

For instance, while initially relying on a consumer-grade GPU (NVIDIA RTX 2070 Super) for computational tasks, it became apparent that its capabilities were insufficient for the scale of this project. Consequently, training was transitioned to a high-performance NVIDIA L4 GPU hosted on a cloud-based server while using the same notebook and environment setup. This switch provided the necessary computational power to handle the extensive training requirements, enabling faster convergence and reducing overall experiment time.

Another critical adaptation involved modifying the game environment itself to define clear objectives for the AI agents. Using Gym Retro, the scenario.json files for both Street Fighter II and Fatal Fury 2 were edited to adjust the episode termination conditions (done). Without this modification, the games would continue playing through their entirety, making it difficult for the agent to focus on mastering specific stages or opponents. The custom condition specifies that an episode ends when either the agent or the opponent wins or loses two rounds (matches). This change ensures that the AI consistently trains to defeat the same opponent in the same stage, aligning the training process with well-defined objectives.

In the event that transfer learning fails to produce a strong policy within an acceptable time frame, further modifications could be implemented. For example:

- Adjusting the learning rate or increasing the number of training timesteps to improve the mean reward.
- Altering the reward function.

5.5 Methodology Assessment

5.5.1 Success

The following requirements must be met in order to be successful:

1. Quicker Training: Compared to the corresponding baseline agents that were trained from scratch, the transfer learning agent should be able to get a solid policy on both the source game and the target game more quickly.
2. Final Performance That Is Equal to or Better: The transfer learning agent should perform at least as well as the corresponding baseline agents in the end on both the source game and the target game.
3. Generalisability: To prove how well the transfer learning framework works in speeding up reinforcement learning for gaming applications, it should be able to be used to a variety of combat games.

These benchmarks and success criteria are in line with the goals of the research, which include creating a workable transfer learning framework for gaming RL and assessing how well transfer learning speeds up RL training in games. When these benchmarks and requirements are met, the suggested strategy will be proven effective, and reinforcement learning for gaming applications will advance as a field.

5.5.2 Evaluation

The evaluation process focuses on achieving the study's objectives through results analysis and comparison:

1. Transfer Learning vs. Baseline Comparison
    - Compare the performance of transfer learning agents to baseline agents across all metrics, including training duration, win rate, and mean reward, mean reward based on specific timesteps and episode length. Assess whether transfer learning significantly accelerates training and achieves comparable or superior performance.
2. Performance Across Games
    - Evaluate the generalisability of the transfer learning framework by testing models trained on one game (Street Fighter II) on a different game (Fatal Fury 2) and vice versa. Assess the retention of learned policies and the ability to adapt effectively to new tasks.
3. Empty-Weight Model Comparison
    - Compare the performance of fully trained models with an untrained, empty-weight PPO model to provide a baseline of how much learning and transfer have improved performance.
4. Usability and Implementation: Assess the transfer learning framework's ease of use, including the effort required to modify the Gym Retro environment, integrate custom reward functions, and train agents. A user-friendly approach ensures its practical utility for future RL research and applications in gaming.

5.6 Safety and Ethical Considerations

There is no safety to consider as all experiments are conducted through software and simulation and require no human interaction. However, there may be ethical considerations in regard to the legality of downloading retro ROMs—Street Fighter II and Fatal Fury 2—for the use of the experiment. The use of ROMs of video games is a legally complex issue. Generally, downloading and using ROMs of video games without proper authorisation from the copyright holders is illegal. However, in this research, the ROMs are used in conjunction with Gym Retro, a platform released by OpenAI specifically for reinforcement learning (RL) research. OpenAI's Gym Retro platform facilitates RL research on a wide variety of games and is designed to support research by providing a diverse set of games from various classic gaming systems, including Sega Genesis, Sega Master System, and Nintendo's NES, SNES, and Game Boy consoles, among others. OpenAI states:

"We're releasing the full version of Gym Retro, a platform for reinforcement learning research on games. This brings our publicly-released game count from around 70 Atari games and 30 Sega games to over 1,000 games across a variety of backing emulators. We use Gym Retro to conduct research on RL algorithms and study generalization." (OpenAI, 2018)

To ensure compliance with legal and ethical standards, only ROMs and emulators provided and supported by OpenAI through Gym Retro will be used in this research. This ensures that the research adheres to the intended use and licensing agreements provided by OpenAI.

6 Results

6.1 Performance Metrics

This section evaluates the performance of three models (M1, M2, and M3) across different scenarios using mean reward and win rate as the primary metrics over 100 games for all evaluations, where each game is an episode. A game or episode is when either the player or opponent wins two rounds out of a best of three. The models were tested in the context of two fighting games, Street Fighter II (SF2) and Fatal Fury 2 (FF2), with a focus on understanding the impact of transfer learning (M3) and the performance of baseline models (M1 and M2).

The first model, M1, was trained from scratch on the Street Fighter II environment for 5 million timesteps, however, the highest mean average by timestep was 3240000 timestep, which was used for evaluating as seen in Figure 11. When tested, M1 achieved a mean reward of 487.09 and a win rate of 83%, demonstrating that it had successfully learned optimal strategies to deplete the opponent's health. These results highlight M1's strong performance in SF2, far surpassing the baseline of a random SF2 model, which achieved a mean reward of 349.8 and a win rate of 52% as illustrated in Figures 12 and 13. Having a random model with a win rate close to 50% is impressive, indicating that the game's default CPU opponent does not pose an overwhelming challenge. However, M1 significantly outperformed this baseline, reflecting its capacity to optimise strategies well beyond what could be achieved without training. The high win rate and reward establish M1 as a strong benchmark for comparison when evaluating the transfer learning effects of M3 on SF2 later.

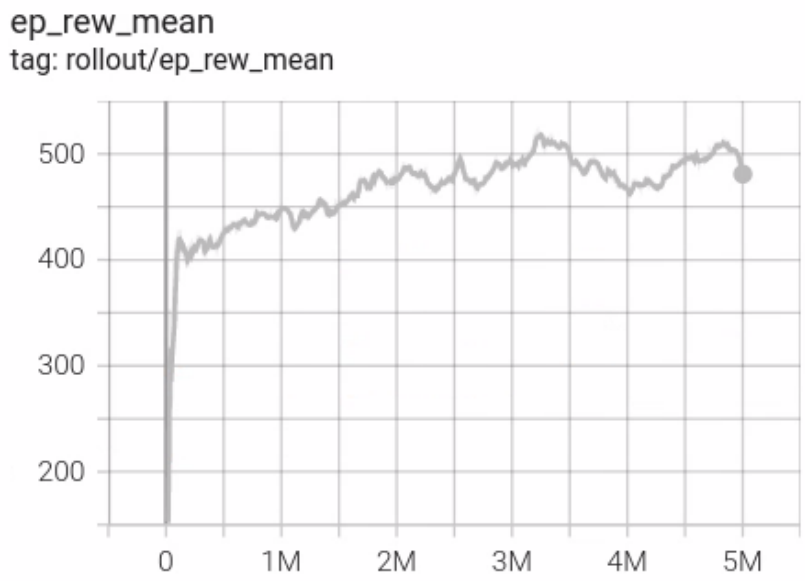

Figure 11: M1 Episode Reward Mean Time Series Graph over 5M Timesteps

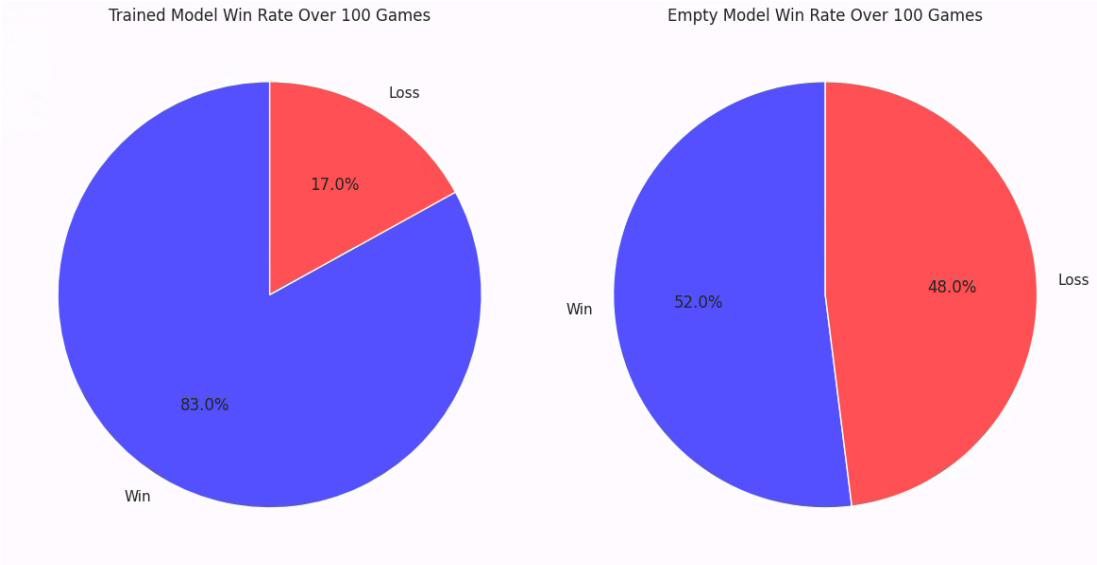

Figure 12: Pie Chart - M1 Win Rate (SF2) against an Empty Model (SF2)

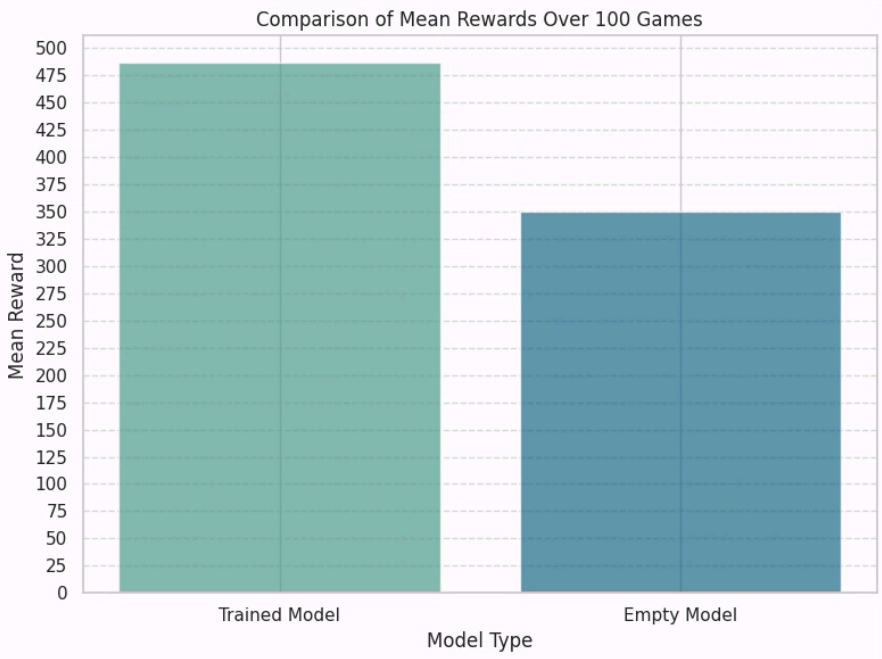

Figure 13: M1 Mean Reward (SF2) against an Empty Model (SF2)

Next, M1 was tested directly on the Fatal Fury 2 environment without any fine-tuning, to evaluate its adaptability to a new game. The mean reward dropped to 341.81, and the win rate decreased to 34%. Despite these reductions, the results were still significantly higher than the random FF2 model, which had a mean reward of 250.45 and a win rate of 7%. This performance indicates that some strategies learned in SF2 transferred to FF2, although suboptimally. The reduced win rate and reward highlight the challenges posed by differences in gameplay dynamics, such as FF2's more aggressive opponents and potentially different move effectiveness. Figure 14 and Figure 19 provide visual representations of these findings.

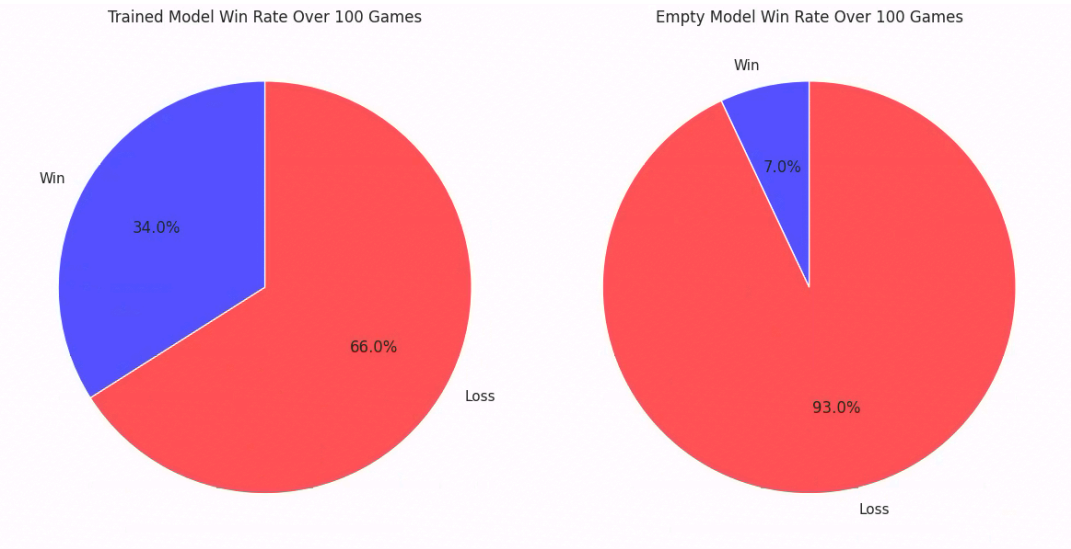

Figure 14: M1 Win Rate (FF2) against an Empty Model (FF2)

To improve performance in FF2, the M1 model was fine-tuned in the FF2 environment, resulting in M3. After 5 million timesteps of fine-tuning as seen in Figure 15 (red coloured line), however, the highest mean average was 3670000 timestep, which was used for evaluating as seen in Figure 15. M3 achieved a mean reward of 344.45 and a win rate of 36%, showing marginal improvements over M1's out-of-the-box results in FF2. These slight gains highlight the potential of transfer learning to refine existing strategies for new environments. However, M3 still underperformed compared to M2, the model trained from scratch in FF2, which achieved a mean reward of 388.59 and a win rate of 39%. Further will be discussed below. This suggests that the fine-tuning period for M3 may have been insufficient for it to fully adapt to FF2's unique gameplay mechanics. These comparisons are visualised in Figure 16 and Figure 19.

Figure 15: M3 Episode Reward Mean Time Series Graph over 5M Timesteps

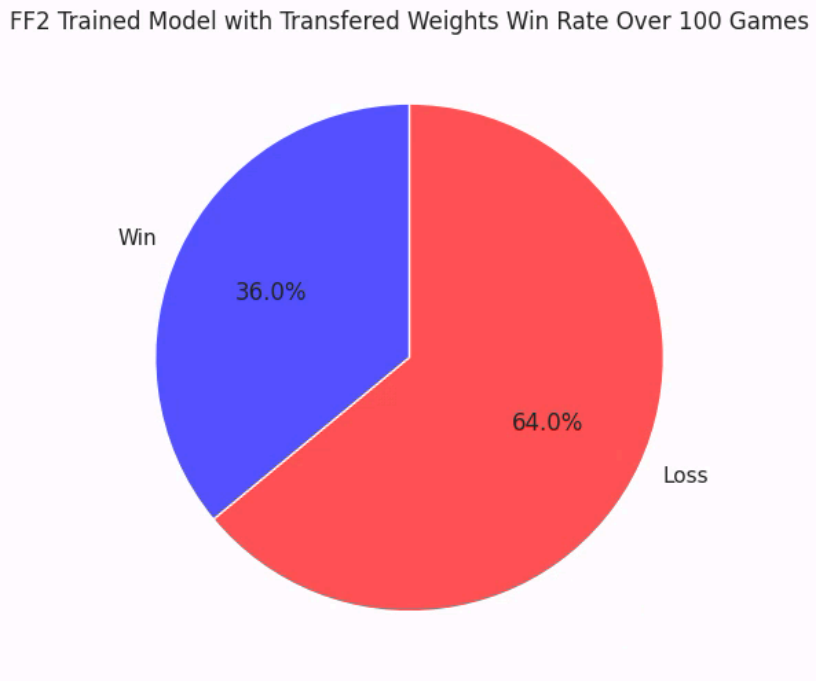

Figure 16: M3 Win Rate (FF2)

The second baseline, M2, was trained exclusively on FF2 for 5 million timesteps, the highest mean average was the full 5000000 timestep, which was used for evaluating as seen in Figure 17 (blue coloured line). It achieved the highest FF2-specific performance, with mean reward and win rate values of 388.59 and 39%, respectively as previously mentioned. This result emphasises the effectiveness of training directly within the FF2 environment, where the model is free to explore strategies without any biases from prior training in SF2. Figure 18 and Figure 19 depict M2's performance in comparison to other models.

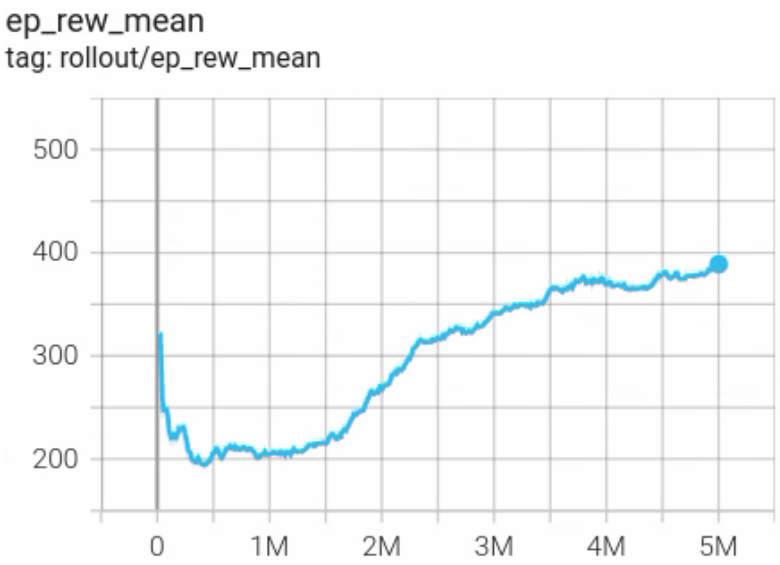

Figure 17: M2 Episode Reward Mean Time Series Graph over 5M Timesteps

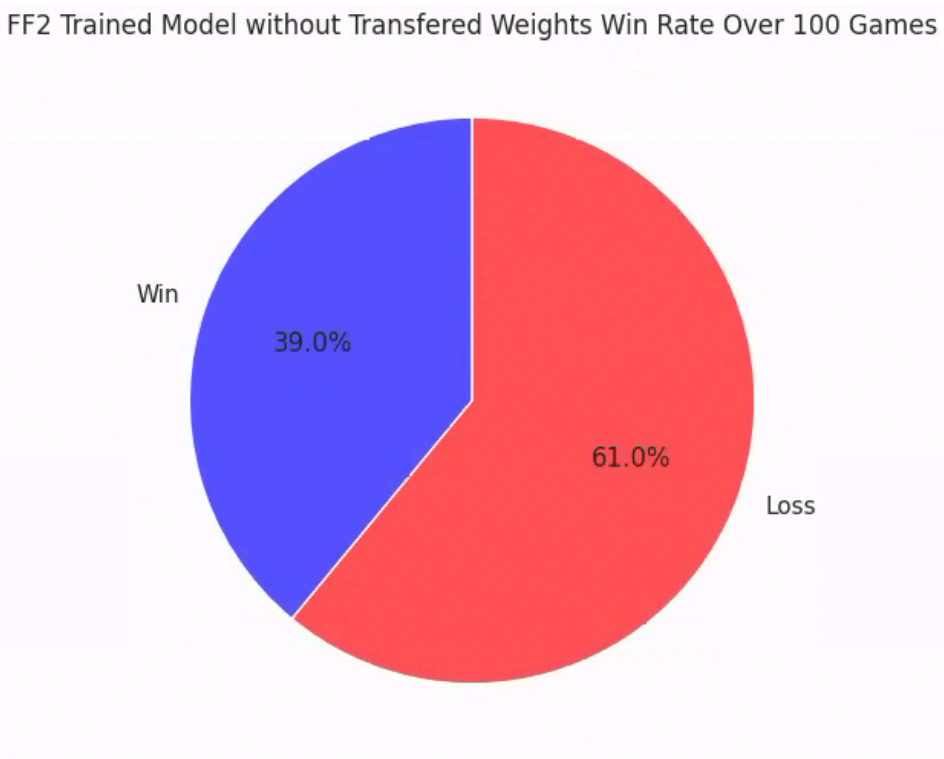

Figure 18: M2 Win Rate (FF2)

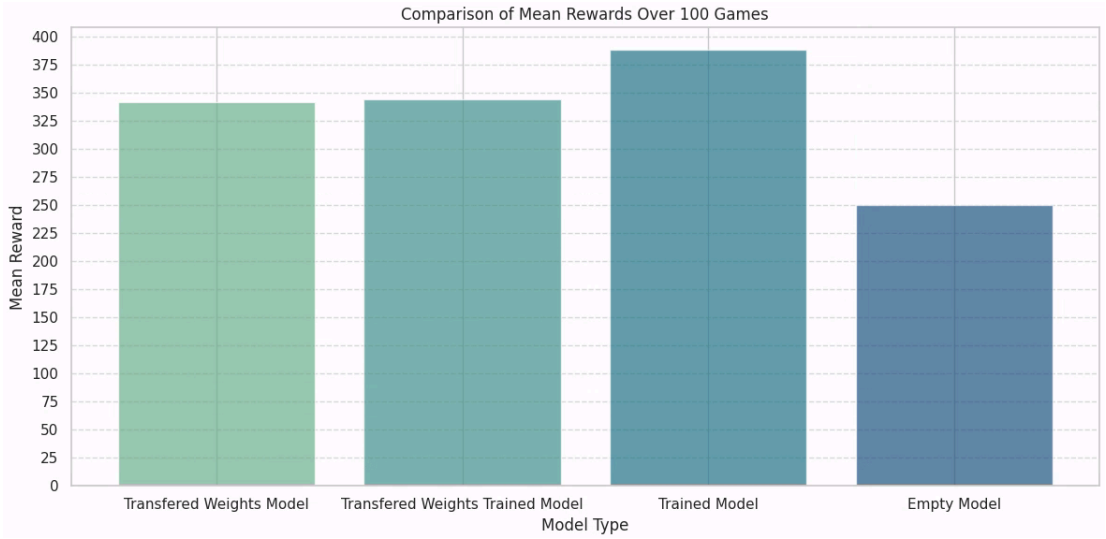

Figure 19: Mean Reward Comparison on Fatal Fury 2

In Figure 19, M1 is Transferred Weights Model, M3 is Transferred Weights Trained Model, M2 is Trained Model and Empty Model is Fatal Fury 2 model with no training.

Finally, M3 was tested again in the Street Fighter II environment to assess whether fine-tuning on FF2 negatively affected its performance in SF. When reintroduced to SF2, M3 achieved a mean reward of 472.67 and a win rate of 77%, slightly lower than M1's original performance. This decline indicates that fine-tuning on FF2 did not lead to catastrophic forgetting but resulted in minor losses in performance. This suggests that while M3 maintained a significant portion of its original SF2 proficiency, further refinement might be needed to preserve peak performance across both environments. These results are illustrated in Figure 20 and Figure 21.

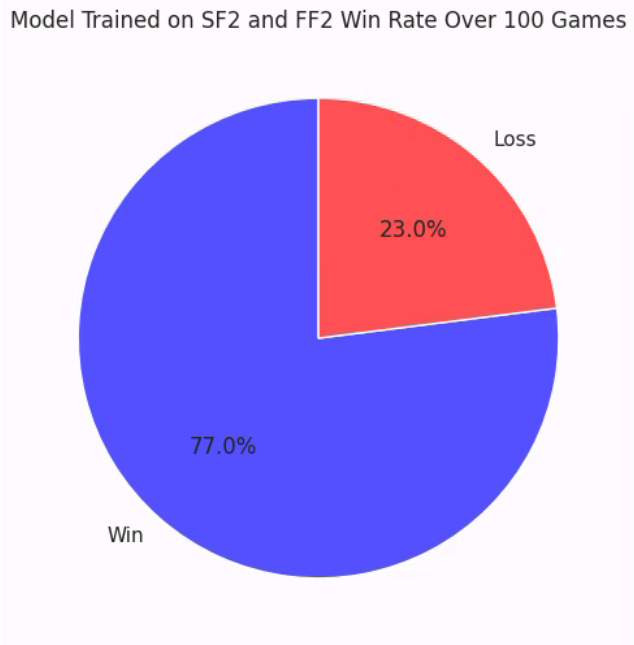

Figure 20: M3 Win Rate (SF2)

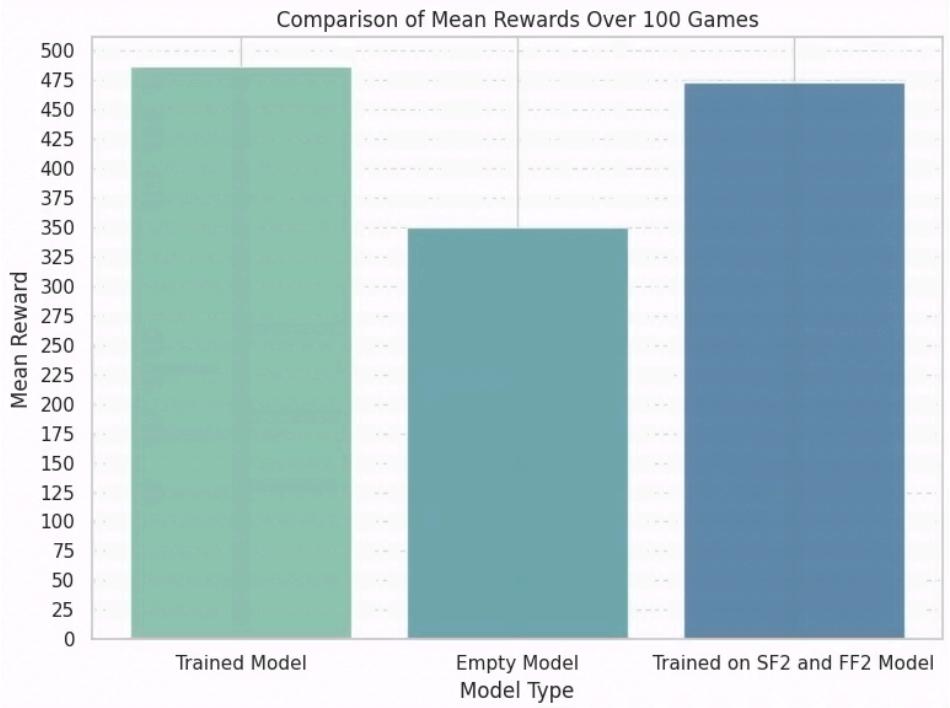

Figure 21: Mean Reward Comparison on Street Fighter II

In summary, M1 demonstrated strong performance in SF2, while M2 showed the best results in FF2 due to dedicated training. Transfer learning with M3 provided moderate improvements in FF2

compared to M1 but did not surpass the performance of M2. Importantly, M3 retained much of its SF2 performance, underscoring the potential of transfer learning to adapt strategies across different environments without significant degradation. These results underscore the promise of transfer learning in reinforcement learning for fighting games, while also highlighting the importance of sufficient fine-tuning to maximise the benefits of this approach.

6.2 Observation Insights

In retrospect, when observing the AI agent and evaluating in action through human render mode, it became evident that the agent would default to crouch-blocking as it was the most optimal way to avoid taking damage. The agent's actions were heavily influenced by frame-by-frame changes made by the enemy, reacting to each frame without a broader strategic vision. This overly reactive approach resulted in the agent holding a defensive position and only throwing a punch or kick immediately after the enemy initiated an attack.

To address this issue in future iterations, a potential improvement would involve implementing a frame delay mechanism in the decision-making process, especially when using deterministic=True in the policy. The AI agent could be trained to delay its actions by a few frames when an enemy performs an attack that changes the frame delta. This would encourage the agent to first block or absorb the initial attack and then respond with a counter-strike, effectively introducing a more strategic, counter-focused playstyle. Such an adjustment would prevent the agent from being locked into its current overly defensive and predictable crouch-blocking behaviour, allowing it to handle a wider range of opponents and progress further in the game.

Figure 22: AI agent crouch-blocking with minimal offensive actions

This behaviour highlighted a fundamental issue with the deterministic evaluation setting (deterministic=True). Under this setting, the agent's policy was evaluated without any randomness, which assumes perfect opponent behaviour. However, given that the enemy's actions were random and unpredictable, a deterministic evaluation setting was suboptimal. To address this, the evaluation setting was toggled to deterministic=False, enabling stochastic evaluation. This introduces randomness into the agent's decision-making process, allowing for exploration during evaluation. Exploration is critical for identifying the best-performing models over multiple trials and refining them further. The implications of deterministic versus stochastic evaluations and their effects on training require further research.

Another issue identified was a declining mean reward trend during training, indicative of suboptimal convergence. The mean reward graph in Figure 23 revealed a downward trend, suggesting the need for adjustments to the reward function and hyperparameters. Initially, the default reward function from the OpenAI Gym Retro settings, which utilised the raw in-game score as the reward, was used. However, this was replaced with a custom reward function that emphasised the delta in the opponent's health as a positive reward and penalised the delta in the agent's health. This adjustment aligned the reward function more closely with the game's objectives. Additionally, training was restricted to a single stage and opponent, allowing the agent to focus on learning effectively rather than spreading its experience thin across multiple stages and opponents. This has alleviated the declining mean reward problem and aimed to accelerate training by providing the agent with more samples for a specific scenario as seen in Figure 24.

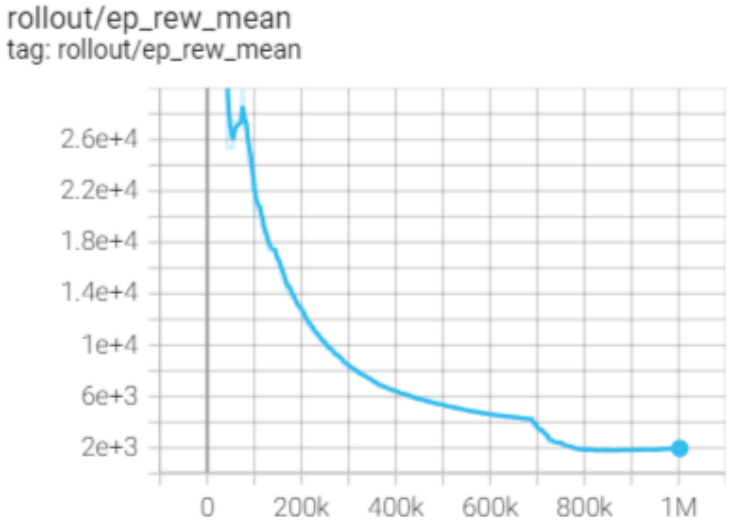

Figure 23: Declining Episode Mean Reward during Street Fighter II Training over 1M Timesteps

As seen in Figure 24, in early training, the initial mean reward graph for 1 million steps showed promising signs. At 100,000 steps, the agent exhibited reduced activity, ultimately converging to a behaviour of blocking with minimal actions. By 500,000 steps, however, the mean reward began to show an upward trend, indicating gradual improvement in performance. This trend suggests potential for the model to converge to an optimal policy with extended training (e.g., 10 to 25 million timesteps) and achieve a mean reward greater than zero.

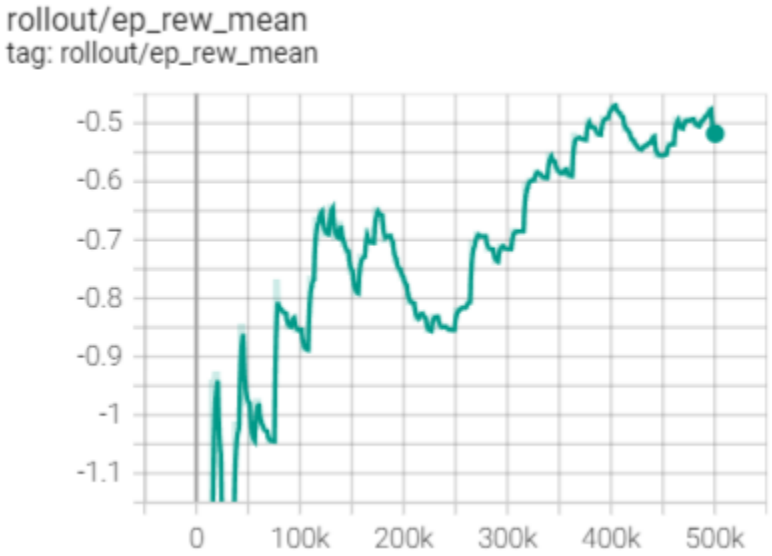

Figure 24: Increasing Mean Reward of SF2 after changing the reward function over 500k Timesteps

For Transfer Learning, when training the model on Fatal Fury 2 with weights transferred from Street Fighter II (M3) compared to training a model from scratch on Fatal Fury 2 (M2), an interesting observation can be made when looking at the mean reward graph. The model with transferred weights significantly outperformed the model trained from scratch until eventually they both broke even at around 2.644M timestep as seen in Figure 25, where the M3 is the red line and M2 is the blue line.

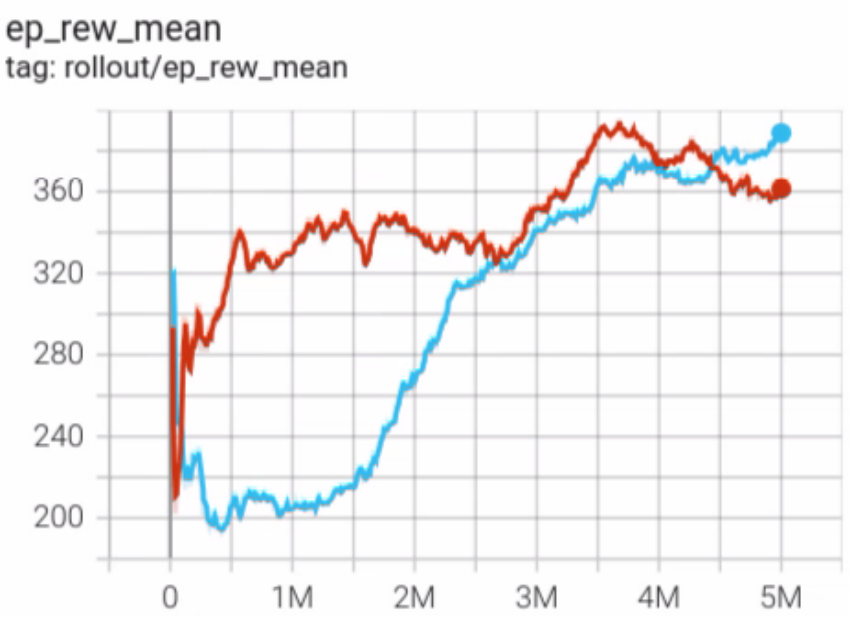

Figure 25: Comparing Episode Mean Reward of M2 (blue) and M3 (red) over 5M Timesteps

A significant claim can be made highlighting that transfer learning does indeed work and does save a significant amount of time, as compared to just training the model from scratch (M2). This can say a few possible things: The model with transferred weights transferred over fundamental game plan, which allowed the ai agent to understand the goals of trying to reduce the enemy's health to zero while trying to avoid taking damage, as set out by the reward function. This claim can be backed up by the episode length mean graph in Figure 26 where the model trained from scratch ends extremely fast as it has no idea on what the goal is and what its game plan is, and is a bit clueless on what it's trying to achieve. In other words, it is losing very quickly but learns to adapt over time and thus seeing the episode mean length increase which eventually breaks even with the model with transferred weights.

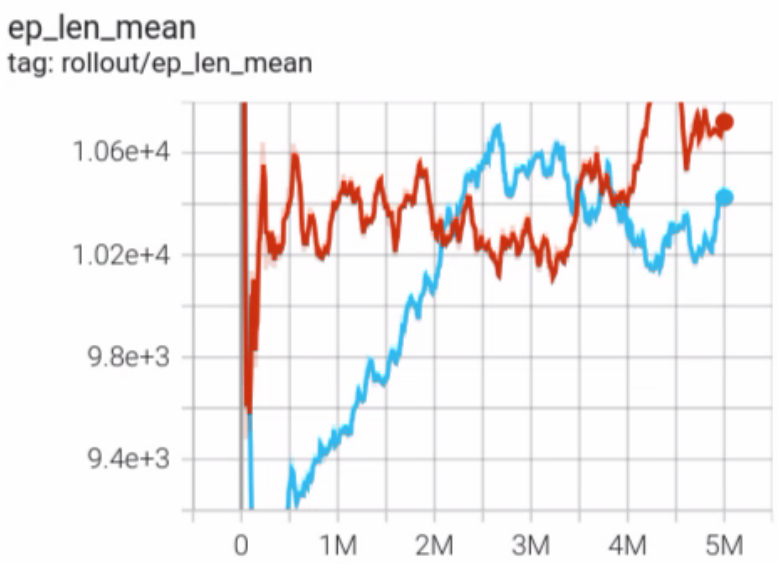

Figure 26: Comparing Episode Length Mean of M2 (blue) and M3 (red) over 5M Timesteps

This can also be seen in the entropy loss between these two, where Entropy in RL measures the "uncertainty" or "randomness" in the agent's action distribution. The entropy loss term encourages exploration by penalising the agent for being too certain in its actions (i.e., overconfidence). A higher entropy means the agent is still exploring a variety of actions, while lower entropy suggests it's more confident in its decisions. In the entropy graph in Figure 27, we can see that for the model trained from scratch, it took a very long time for it to spike or “start exploring” compared to the model with transferred weights since, that model already know the game play, so all it is doing now is trying to figure a way to beat a new opponent.

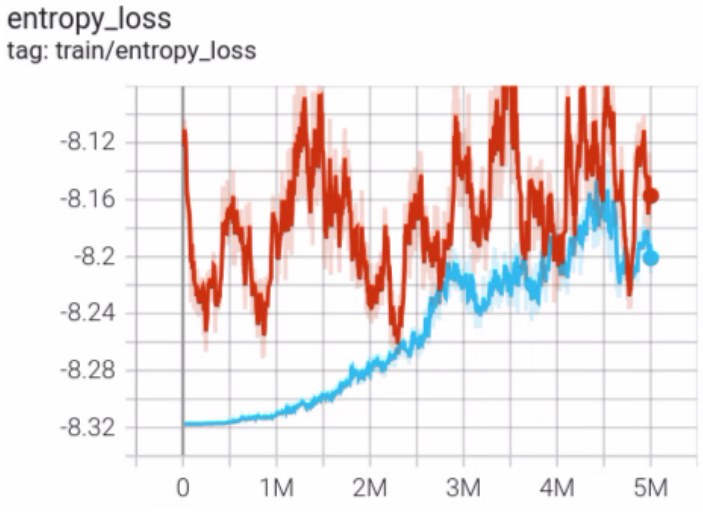

Figure 27: Comparing Entropy Loss of M2 (blue) and M3 (red) over 5M Timesteps

7 Discussion

7.1 Deterministic vs Stochastic Approach

The only way the model can achieve a 100% win rate is by evaluating using a deterministic approach. This would allow the AI to possibly find a strategy or loophole that it can exploit in the game bot, often leading to consistent victories and potentially the maximum win rate. However, due to the nature of Street Fighter II being multibinary with complex and numerous button combinations, movements, and actions, having the AI agent perform the same moves repeatedly will not be viable in the long run. Such an approach can be exploited if the AI agent encounters stronger opponents. Therefore, making the bot play with a stochastic approach better suits the genre of fighting games, where opponents do not rely on repetitive moves because of inherent randomness in real-world games. This introduces variability, making it more adaptable. Consequently, evaluating the convergence of the model becomes challenging, as the AI agent is unlikely to repeat the same actions during evaluation due to the probability-driven nature of a stochastic network. This inconsistency complicates determining convergence during training.

Additionally, the complexity of the multibinary action space means there are too many possible scenarios. For example, at 5 million timesteps, there may still not be enough data to cover all combinations of scenarios, frames, and pixels. After a certain number of timesteps—in this case, 3,240,000—the performance started to decline. A quick evaluation of the model at 3,240,000 timesteps compared to the model at 5 million timesteps showed that the earlier model outperformed the later one in both mean reward and win rate. This could be due to several reasons:

1. Overfitting to the Training Environment:

Over time, the model may overfit to specific patterns or characteristics of the training environment. This overfitting could cause the AI to exploit environment-specific weaknesses or become too specialised in a narrow set of strategies, reducing its ability to generalise to new or unseen scenarios. This is reflected in a decline in performance metrics such as win rate or reward.

2. Policy Collapse (Exploration Issues):

Reinforcement learning algorithms like PPO rely on a balance between exploration (trying new actions) and exploitation (choosing the best-known action). Over extended training periods, the agent may reduce its exploration too much and settle into suboptimal strategies—a phenomenon called policy collapse. If mechanisms like entropy regularisation or the learning rate decay too quickly, the model stagnates, preventing further improvement and causing performance to degrade.

3. Vanishing/Exploding Gradients or Optimization Issues:

Prolonged training can sometimes lead to optimization issues like vanishing or exploding gradients, which make the learning process unstable. These issues can result in the model diverging from an optimal policy. Improper learning rate adjustments or overly aggressive updates to the policy can exacerbate this problem, ultimately hindering performance.

7.2 Game Difficulty and its Effect on Training

Both Street Fighter II and Fatal Fury 2 are complex games with extensive action spaces, each having hundreds of valid button combinations due to being MultiBinary(12). These games require extensive exploration and prolonged training to achieve optimal performance—potentially around 25 million timesteps, or even in the billions. However, one factor that either eases or complicates training is the difficulty of the opponent.

For the Street Fighter II model, it was observed that the AI did not require extensive training to achieve a high win rate of 83%. Even without training, the AI had an impressive initial win rate of approximately 50%. This performance is likely influenced by the nature of the opponent and the game settings. For instance, the AI agent faced "Guile," who, based on observation, was not overly aggressive and did not block every attack. Furthermore, specific moves, such as Ryu's tatsumaki senpukyaku (tatsu), provide a significant advantage. This move has projectile invincibility, allowing the AI agent to counter Guile's projectile-based attacks effectively. If the AI learns to utilise this move or spam it, it can efficiently counter Guile and improve training performance.

In contrast, training on Fatal Fury 2 posed greater challenges. The AI agent's mean reward remained in the negatives for a prolonged period, highlighting the increased difficulty of the opponent. Unlike Guile in Street Fighter, the opponent in Fatal Fury 2 exhibited more aggressive and challenging behaviour, making it harder for the AI to develop effective strategies. Additionally, Terry, the character used in Fatal Fury, does not have access to a move equivalent to Ryu's tatsu, which inherently limits the AI's aggression and increases its vulnerability to attacks. These factors slowed down training and reduced performance consistency.

Measuring the difficulty of opponents across both games remains challenging due to the lack of access to game settings and CPU algorithms. As a workaround, extending the training duration became necessary to allow the AI agent more time to adapt to the complexities of the environment. Despite

these limitations, the observations underscore the significance of opponent difficulty and game-specific mechanics in shaping the training and performance of reinforcement learning models.

7.3 Game Similarity and Character Archetype

In Street Fighter II and Fatal Fury 2, the AI agent controls the characters Ryu and Terry, respectively, which the Fighting Game Community (FGC) categorises under the archetype known as a "Shoto." According to the Fighting Game Glossary, a Shoto is "an archetype in the Street Fighter series for a character that has a fireball, a shoryuken, and a tatsu." This means that both characters share very similar movesets and button combinations. For instance, the input for a fireball move—Down, Down Right, Right, Punch—is the same for both characters.

Since the button layout and MultiBinary action space are identical across the two games, the model trained on Street Fighter II is capable of effectively transferring its knowledge to Fatal Fury. The model "understands" the button combinations required for special moves in Street Fighter II and can apply the same inputs in Fatal Fury 2 because the button mapping remains consistent.

However, despite the similarity in movesets, the environment's setup, where the AI agent perceives frame-by-frame observations, poses challenges. For example, when training the model on Fatal Fury, the agent initially does not "know" when to execute these learned motions, even though it has already acquired them from Street Fighter II. That said, the performance data demonstrates that transfer learning still positively impacts the model. Training the model on Street Fighter II weights improved performance during Fatal Fury training, as the agent adopted a more aggressive playstyle, leading to higher mean rewards.

While transfer learning does not negatively affect the model's learning, it also did not allow the agent to converge to a significantly higher mean reward within the 5 million timesteps available for training. Nevertheless, the mean reward shows a consistent upward trend, suggesting that with more training and additional timesteps, the model's performance in terms of win rate and mean reward would likely improve further. Unfortunately, due to time constraints, I was unable to train the models beyond 5 million timesteps.

One key observation is that the aggression learned during training on Street Fighter II successfully transferred to Fatal Fury. When loading the Street Fighter model weights into Fatal Fury without further training, the agent was able to perform comparably to a model that had been specifically

trained on Fatal Fury. This suggests that the foundational gameplay knowledge and aggressive playstyle were successfully retained and applied across games, even with limited training.

7.4 Testing the Convergence

Testing convergence was extremely limited in this study due to time constraints, limited GPU resources, and strict deadlines. As a result, I was only able to select models from specific timesteps that exhibited the highest mean reward and use them as final models. This approach, while practical under time limitations, is not the most effective method for studying true convergence. In the future, to properly assess whether the model can achieve convergence and how quickly it does so, significantly longer training durations would be required—potentially exceeding 1 billion timesteps. This would allow the model to adequately explore the extensive multibinary action space, encounter a greater variety of scenarios, and refine its strategies more effectively.

Instead of directly testing convergence, I focused on evaluating the performance of the model by analysing its win rate and mean reward over a set number of games. I also attempted to estimate the training time required for the model to peak in terms of mean reward. While this provided some insights, it falls short of rigorously demonstrating convergence behaviour. For context, the original Atari paper (Mnih et al., 2013) tested convergence over 10 million timesteps for environments that are comparatively simple in terms of mechanics and action space. For more complex environments such as Street Fighter or Fatal Fury, which feature numerous possible inputs and highly dynamic gameplay, achieving convergence would require significantly longer training durations. For example in the paper *The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games*, where the domains trained were highly complex game environments including StarCraft II, all the methods were trained for at-most 10 billion environment steps (Yu et al., 2022) which sees the model converge.

Convergence in reinforcement learning (RL) can be assessed by monitoring several key metrics over time, such as mean reward, policy entropy, and the standard deviation of rewards. If these metrics stabilise, it may indicate convergence. However, convergence does not necessarily imply optimal performance. A model can stabilise at a suboptimal policy, particularly in complex environments like fighting games.

Key challenges in testing convergence for this study include:

1. Environmental Complexity:

Both Street Fighter and Fatal Fury have intricate environments with high-dimensional inputs (e.g., pixel frames) and complex action spaces. This increases the number of scenarios the agent must learn, requiring significantly more timesteps to achieve convergence.

2. Exploration vs. Exploitation Balance:

The agent must strike a balance between exploring new strategies and exploiting known ones. If the agent over explores, it may fail to refine a strong strategy. Conversely, if it over exploits too early, it risks getting stuck in suboptimal behaviour. Monitoring entropy regularisation and adjusting it dynamically could improve convergence.

3. Plateauing Performance:

As observed during training, the agent's performance can plateau or even decline after certain timesteps due to issues like overfitting, insufficient exploration, or suboptimal hyperparameter tuning. Adaptive methods, such as learning rate scheduling or dynamic entropy bonuses, could help the agent overcome these plateaus.

4. Time Constraints:

With more time and computational resources, longer training runs would provide a clearer picture of convergence. Additional experiments, such as varying the number of training episodes, fine-tuning hyperparameters, or testing alternative architectures (e.g., transformers), could help refine the training process.

7.5 Future Works

Based on the results of this study and limitations of experimentation, some directions can be suggested for future research to improve the training, evaluation, and performance of reinforcement learning models in complex environments like fighting games.

7.5.1 Addressing Training Time

One of the most significant challenges faced in this study was the limited training time. Future work could explore using more advanced hardware, such as higher-end GPUs or even distributed computing systems, to enable training for significantly longer durations—up to 25 million timesteps or more. This would allow the model to better explore the action space, encounter more diverse scenarios, and develop a deeper understanding of optimal strategies. Additionally, implementing dynamic hyperparameter adjustments during training, such as scheduling learning rates or tweaking entropy

regularisation, could help the model train more efficiently and avoid issues like policy collapse or plateauing performance.

7.5.2 Expanding Evaluation

While this study demonstrated some promising results, the evaluation was limited to specific games and controlled environments. Future research could test the transferability of trained models across a wider variety of games with different mechanics and styles, to see how well the model generalises beyond similar action spaces. Another exciting direction would involve testing the model against human players or in multi-agent environments, which would offer a better understanding of its real-world applicability and ability to adapt to unpredictable opponents.

7.5.3 Developing a More Complex Reward Function

The reward function was relatively simple in this work: rewards and penalties mainly consisted of changes in player and enemy health. But in further research, the reward function can become more complex and also include other elements, like the number of successful combos, the efficiency of movement, or the rate of defensive strategies blocking. With the agent, this can also involve providing it with more detailed feedback, as it may improve performance and lower the overall training time needed for convergence. Experiments with reward structure could also help in making the agent learn the nuances of fighting game mechanics better.

7.5.4 Generalisation of Transfer Learning for Future Research

This study demonstrated that transfer learning can be effective in complex environments like fighting games, provided the environments share similar goals, mechanics, and controls. In this case, the model trained on Street Fighter successfully transferred its knowledge to Fatal Fury due to the shared objective of depleting the opponent's health to zero across two rounds to achieve victory. The agent retained an aggressive game plan developed during Street Fighter training, which proved effective in Fatal Fury. This highlights the potential of transfer learning as a framework for applications where environments align in key aspects.

However, this success also underscores certain limitations and areas for improvement. As observed in this experiment, transfer learning worked effectively because both games shared similar reward structures, action spaces, and objectives. For transfer learning to generalise to more diverse environments, several challenges must be addressed:

Building Frameworks for Broader Use Cases

Future work could explore transfer learning frameworks designed for environments that differ more significantly in goals or mechanics. For example, adapting a fighting game model to a platformer or strategy game would require more sophisticated methods for mapping knowledge between vastly different environments. Developing methods for identifying transferable components of a policy—such as high-level strategies, control patterns, or reward structures—would be a crucial step toward generalisation.

Expanding Beyond Similar Reward Functions

Currently, transfer learning is most effective when the reward functions of the source and target environments align. In this study, both games used health depletion as the primary goal, allowing the agent's aggressive strategy to remain viable. Future studies could experiment with environments where goals differ—such as games requiring defensive play, resource management, or navigation. This would require creating adaptable reward functions or multi-objective frameworks to enable the agent to transfer core knowledge while adjusting to new objectives.

Improving Performance Through Extended Training

While transfer learning successfully preserved the aggressive playstyle between games, achieving optimal performance in the target environment still requires substantial additional training. To enhance the generalisation capability of transferred models, longer training durations and more comprehensive exploration of the target environment would be necessary. Advanced techniques like fine-tuning, domain adaptation, or dynamic re-training could further improve performance in the target environment while leveraging knowledge from the source.

Standardising Controls and Game Plans

As this experiment demonstrated, transfer learning is more likely to succeed when the controls and game plans of both environments are similar. For broader applicability, research could focus on developing standardised representations of control schemes and strategic behaviours, enabling smoother transitions between environments with differing input mechanisms. Tools like embedding techniques or control abstraction layers could help bridge this gap.

8 Conclusion

This study set out to explore whether transfer learning could be used as a generalisable framework to improve the training time, performance, and convergence rate of deep reinforcement learning (DRL)

agents. By applying transfer learning to two fighting games—Street Fighter II and Fatal Fury 2—the results demonstrate that transfer learning is not only feasible but also effective in reducing training time and enhancing agent performance when the source and target environments share similar objectives, mechanics, and control schemes. The findings indicate that transfer learning enabled the agent to retain a fundamental game plan, such as aggression, learned from Street Fighter II and successfully apply it in Fatal Fury 2. This allowed the model to perform competitively in the target game with minimal initial training. However, achieving optimal performance still required additional fine-tuning, highlighting that while transfer learning can provide a strong starting point, further training is necessary to fully adapt to new environments.

The research question—"Can transfer learning be used as a generalisable framework to improve the training time, performance, and convergence rate of DRL agents?"—is supported, but with important caveats. Generalisation is contingent on the alignment of critical factors such as reward functions, control mappings, and shared gameplay strategies. The experiments showed that when these elements are consistent between environments, transfer learning accelerates convergence and enhances sample efficiency. However, for broader applicability, additional work is needed to handle cases where environments differ significantly in goals or mechanics.

This study contributes to the growing body of research on sample-efficient reinforcement learning by demonstrating that transfer learning can be an effective tool for training DRL agents in complex, dynamic environments like fighting games. It offers a pathway for extending RL applications to more resource-intensive domains where training from scratch is impractical. Future work can build upon these findings by addressing the generalisation of transfer learning to more diverse environments, refining reward structures, and improving training methodologies to further optimise performance and convergence rates.

In conclusion, while this research confirms the viability of transfer learning as a framework for improving DRL training, it also highlights the importance of tailoring transfer strategies to the specific characteristics of source and target tasks. By addressing these challenges, transfer learning holds the potential to significantly advance reinforcement learning applications in gaming and beyond, offering solutions to complex real-world problems that demand adaptability, efficiency, and resilient decision-making.

9 Acknowledgements

I would like to thank my supervisor Mingshan Jia for providing access to use UTS iHPC and support with constant meetings and suggestions throughout this project.

References

[1] Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019, July 25). *Optuna: A next-generation hyperparameter optimization framework*. arXiv.org. https://arxiv.org/abs/1907.10902

[2] Bellman, R. (1957). *A Markovian Decision Process*. Indiana University Mathematics Journal, *6*(4), 679–684. https://doi.org/10.1512/iumj.1957.6.56038

[3] He, G., Yang, D., & Shen, K. (2020). *Improving DQN Training Routines with Transfer Learning*. Stanford University. https://georgehe.me/dqn_transfer.pdf

[4] Hsu, S.-H., Shen, I.-C., & Chen, B.-Y. (2018, September 4). *Transferring deep reinforcement learning with adversarial objective and augmentation*. arXiv. https://doi.org/10.48550/arXiv.1809.00770

[5] Li, Y. (2018, October 15). *Deep Reinforcement Learning*. arxiv.org. https://arxiv.org/abs/1810.06339

[6] Moos, J., Hansel, K., Abdulsamad, H., Stark, S., Clever, D., & Peters, J. (2022, March 19). *Robust reinforcement learning: A review of foundations and recent advances*. MDPI. https://doi.org/10.3390/make4010013

[7] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013, December 19). Playing Atari with Deep Reinforcement Learning. ArXiv.org. https://arxiv.org/abs/1312.5602

[8] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015, February 25). *Human-level control through deep reinforcement learning*. Nature News. https://www.nature.com/articles/nature14236

[9] OpenAI (2018, May 25). *Gym Retro*. OpenAI. https://openai.com/index/gym-retro/

[10] OpenAI, Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., Pinto, H. P. d. O., Raiman, J., Salimans, T., ... Zhang, S. (2019, December 13). *Dota 2 with large scale deep reinforcement learning*. arXiv. https://doi.org/10.48550/arXiv.1912.06680

[11] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., & Dormann, N. (2021, November 21). *Stable-baselines3: Reliable reinforcement learning implementations*. Journal of Machine Learning Research. https://www.jmlr.org/papers/v22/20-1364.html

[12] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017, August 28). Proximal Policy Optimization Algorithms. ArXiv.org. https://arxiv.org/abs/1707.06347

[13] Sutton, R. S., & Barto, A. G. (2022, October 20). *Reinforcement Learning*. MIT Press. https://mitpress.mit.edu/9780262039246/reinforcement-learning/

[14] Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., & Liu, C. (2018, August 6). A survey on Deep Transfer Learning. arXiv.org. https://arxiv.org/abs/1808.01974

[15] Yu, C., Velu, A., Vinitsky, E., Gao, J., Wang, Y., Bayen, A., & Wu, Y. (2022, November 4). The Surprising Effectiveness of PPO in Cooperative, Multi-Agent Games. ArXiv.org. https://doi.org/10.48550/arXiv.2103.01955

[16] Zhu, Z., Lin, K., Jain, A. K., & Zhou, J. (2023, July 4). *Transfer Learning in Deep Reinforcement Learning: A Survey*. IEEE Xplore. https://ieeexplore.ieee.org/document/10172347/

[17] Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., & He, Q. (2020, June 23). *A comprehensive survey on Transfer Learning*. arXiv.org. https://arxiv.org/abs/1911.02685