universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Representative Subset Selection from RL Benchmarks

verfasst von | submitted by

Varun Devgon

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2024

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 921
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Informatik
Betreut von | Supervisor:	Assoz. Prof. Dipl.-Ing. Dr.techn. Sebastian Tschiatschek BSc

Acknowledgment

I express my sincere gratitude to my advisor, Timo Klein, B.Sc. M.Sc., for his insightful guidance and always being so patient, positive, and encouraging throughout my thesis research. Our bi-weekly meetings, where we discussed my progress, hurdles, and redirection, have been an invaluable source of support.

Abstract

This proof of concept research aims to identify representative environments from multiple Reinforcement Learning benchmarks, enabling efficient multi-benchmark testing to develop more generalizable and robust Reinforcement Learning algorithms. The methodology proposed in this thesis builds upon the ideas presented in the Atari-5 paper to determine the most representative three environment subset for a combined benchmark of 38 environments, which includes 26 Atari100k environments and 12 DeepMind Control environments. This is achieved using a new normalization technique that uses the performance of a random agent as a baseline and a PPO agent's score at convergence as a reference, followed by a log transformation. This is performed to make the algorithm scores across different benchmarks comparable. Linear regression models from benchmark subsets are then created to predict full benchmark weighted median scores across all algorithms. The results reveal that the three environments Ms. Pacman, Ball in Cup Catch, and Pendulum Swingup, successfully predict algorithm median scores with a relative error of only 6.59% at only 7.9% of computational costs compared to the full benchmark of 38 environments.

Kurzfassung

Das ist eine deutsche Kurzfassung meiner in Englisch verfassten Masterarbeit.

Diese Proof-of-Concept-Forschung zielt darauf ab, repräsentative environments aus mehreren Reinforcement Learning Benchmarks zu identifizieren, um effizientes Multi-Benchmark-Testing zu ermöglichen und damit allgemeinere und robustere Reinforcement Learning Algorithmen zu entwickeln. Die in dieser Arbeit vorgeschlagene Methodik baut auf den Ideen aus dem Atari-5-Papier auf, um das repräsentativste drei Environment-Subset für einen kombinierten Benchmark von 38 environments zu bestimmen, die 26 Atari100k environments und 12 DeepMind Control environments umfasst. Dies wird durch eine neue Normalisierungstechnik erreicht, die die Leistung eines zufälligen Agenten als Basislinie und den score eines PPO-Agenten bei Konvergenz als Referenz verwendet, gefolgt von einer logarithmischen Transformation. Dies geschieht, um die algorithm scores über verschiedene Benchmarks vergleichbar zu machen. Anschließend werden lineare Regressionsmodelle aus den Benchmark-Teilsets erstellt, um die gewichteten Medianscores des vollständigen Benchmarks über alle Algorithmen vorherzusagen. Die Ergebnisse zeigen, dass die drei environments - Ms. Pacman, Ball in Cup Catch und Pendulum Swingup - erfolgreich die Medianscores der Algorithmen mit einem relativen Fehler von nur 6.59% vorhersagen, bei nur 7.9% der Rechenkosten im Vergleich zum vollständigen Benchmark von 38 environments.

Contents

1 Introduction

Reinforcement learning (RL) [33] is a trial-and-error-based machine learning method that improves an agent's policy in navigating its environment. RL has a wide range of applications in fields like robotics and gaming particularly in autonomous navigation, game playing and testing. Model-free RL optimizes the agent's policy solely by interacting with the environment and learning from rewards [33]. Various RL algorithms can be used to find an optimal policy. Generally, the usefulness of these algorithms is tested by measuring their performance using standardized benchmarks to judge how well an algorithm can generalize across different environments. Some popular example benchmarks in this context are the Atari [3], Procgen [6], DeepMind Control [35] and Minihack[27]/Nethack[19] environment suites. Since these benchmarks contain various environments and tasks, they test these algorithms under many conditions. Understandably, the performance can differ among different environments, even within the same benchmark.

Reinforcement Learning algorithms are developed, trained, and compared using standardized benchmarks, which consist of RL environments with different properties and complexities. However, training high-performing deep RL agents often takes substantial time, making using multiple benchmarks computationally difficult [13]. This often leads to RL algorithms generalizing poorly to novel environments, potentially due to over-fitting on a specific benchmark or its subset [17]. Since RL is slow and resource-consuming, training an agent using an algorithm on even one full benchmark is very expensive. This becomes an even greater challenge when considering multiple benchmarks. Additionally, hyper-parameter settings and the influence of randomness can significantly alter the performance, making it difficult to judge if the trained model is superior to baselines or only over-fitted to the tested benchmark or random seed [13, 18].

I aim to fill this gap by selecting a representative subset from multiple RL benchmarks

combined to reduce computational costs and improve the feasibility of multi-benchmark algorithm testing. This contributes to a direction in RL research that does not heavily rely on expensive benchmark testing but, instead, focuses on obtaining reliably extrapolated results from a meaningful subset of full benchmarks at fractional compute costs [1].

In this proof of concept thesis, I proposed a methodology which found a representative subset of environments across a 38 environment combined benchmark consisting of 26 environments from Atari100k [16] and 12 environments from DeepMind Control [35] - to reliably predict the overall benchmark performance. I found the three environments Ms Pacman, Ball in cup catch, Pendulum swingup - which closely estimated the median algorithm performances from the full benchmark with 6.59% relative error using only 7.9% computational costs in comparison to the full benchmark.

In section 2, I give a brief introduction to the basic concepts of Reinforcement Learning, RL environments, algorithms and the implementation libraries relevant to my experiments. In the next section 3, I discuss the relevant research papers and recent successful results which inspired the idea for identifying representative environments from multiple benchmarks to enable efficient multi-benchmark testing. Next, in the section 4, I present my research question with an overview of my research concept, planned approach, challenges encountered and the adjustments made. Further, in section 5, I present the implementation steps for my conducted experiments to find the most representative subsets from Atari100k [16] and DM Control1m [35] benchmarks. Finally, in sections 6 & 7, I present the results in the form of the best subset discovered followed by some evaluation experiments and a case study. Then in the section 8, I discuss the limitations of my research along with the ideas of future work. Lastly, in section 9, I briefly conclude the findings of this thesis.

2 Background

This section gives a brief background to the basic concepts of Reinforcement Learning (RL) including the RL environments, algorithms and the implementation libraries relevant to this thesis.

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a sequential decision-making framework that learns through feedback from an agent interacting with its environment. Using this knowledge, it aims to obtain an optimal policy, which is a decision rule that enables the agent to choose the best action that can be taken from any state. RL solves this problem by maximizing the cumulative reward at the end of an episode or after certain time steps [33, 31], taking into account the exploration/exploitation dilemma.

The compromise between choosing high reward returning actions from a state (exploitation) and discovering new actions which return better rewards (exploration) is crucial in finding the optimal policy [33, 31]. This balance is particularly important when rewards are sparsely distributed in an environment, as exploring new states ensure that the agent can find those rare rewards to improve the policy.

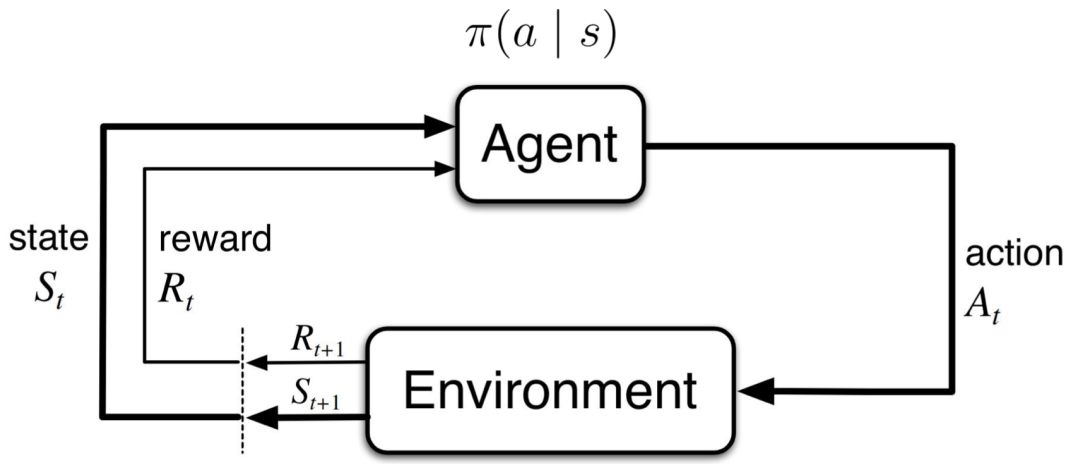


Figure 1: RL loop - source [33, 18]. An agent at state S and time t defined as S_t follows policy π to take an action A_t. The environment moves the agent to its new state S_{t+1} while returning a reward R_t.

Markov Property and Decision Process

Markov Property The conditional probability of transitioning to the next state S_{t+1} given the current state S_t is the same as the conditional probability of the next state S_{t+1} given all the states until S_t [31, 33]. Thus, the current state S_t captures all the relevant information from the history of $S_1, \ldots, S_t$ states.

Markov Decision Process or an MDP is a mathematical framework that adheres to the Markov property and lays a formal foundation for Reinforcement Learning [33].

It is formally written as a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ [33] for an environment, where $\mathcal{S}$ is the set of states - the possible positions where the agent can be in its environment. $\mathcal{A}$ is the set of actions - the possible actions (discrete or continuous) that an agent can take from a given state. P is the state transition probability matrix representing the conditional probabilities $P(S_{t+1} \mid S_t, A_t)$ of an agent at a state S_t, taking an action A_t

and transitioning to a new state S_{t+1}. $R(S_t, A_t, S_{t+1})$ is the reward function - returns a scalar reward feedback as a response to an agent's current state S_t, taken action A_t and the new state after transition S_{t+1}. $\gamma \in [0, 1]$ is the discount factor, a parameter controlling the farsightedness of the agent. With $\gamma = 0$, the agent is totally myopic, only caring about the immediate reward and disregarding future rewards. At $\gamma = 1$, rewards are given equal importance irrespective of whether they are received immediately or at time step ∞. The MDP is formalized and practically realized as RL environments. RL environments are explained in the following section.

2.2 RL Environments

. An RL Environment is a simulated world that the agent interacts with through actions $\mathcal{A}$. The environment, in response, gives the agent a real-valued scalar feedback $R_t = R(S_t, A_t, S_{t+1})$ and moves the agent from S_t to S_{t+1} [33]. The internal dynamics of the environment may or may not be hidden to the interacting agent. There can be many different environments like text-based, gridworlds (e.g., frozenLake) [5], classic control (e.g., cartpole, pendulum) [5], DeepMind Control [35], classical or modern games etc.

Two key properties of RL training environments are Observation/State space and Action space:

Observation & State space An observation is a partially informative representation of what is visible to an agent at a given time step. In contrast, a state contains all information of the environment, including both observable and unobservable variables at a given time step [33]. The set of all possible observations and states for a given environment defines the environment's observation and state space, respectively. The observation and state spaces can be discrete or continuous depending on the environment. For example, the position of the agent in the OpenAI Gym Cliff Walking [5] for the

discrete case and the position and velocity of the car in OpenAI Gym Mountain Car environment [5] for the continuous case.

Action space The set of possible actions an agent can take from any state defines the environment's action space [33]. The action space can be both discrete or continuous, depending on the environment. For example: OpenAI Gym FrozenLake [5] environment has a discrete action space with actions like LEFT, RIGHT, UP, DOWN to move the agent in a gridworld environment. Whereas the DM Control Ant [35] has a continuous action space where actions have precise real values controlling the torque and force applied to the ant's legs to move forward.
Some of the most popular RL benchmarks which offer a variety of discrete and continuous control environments are OpenAI Gym [5] & Frama foundation Gymnasium [38], Procgen [6], DM Control [35], Minihack [27], Atari [3] etc.

2.3 RL Algorithms

RL algorithms provide a framework for an agent to learn an optimal policy by iteratively improving its initially suboptimal decision rule based on the rewards received from the environment [31].

Two key concepts in RL algorithms are the State and Action Values:

State Value or V value [33] gives the expected cumulative reward for an agent at state S_t if it follows a policy π from that state. The V values gives the favorability of different states for an agent in the environment [31].

$$V_\pi(S_t) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_t \right] \tag{1}$$

where $V_\pi(S_t)$ denotes the state-value function, which represents the expected cumulative reward starting from state S_t and following policy π. Similarly, $\mathbb{E}_\pi$ denotes the expectation over the sequence of actions and states generated by following policy π. The summation $\sum_{t=0}^{\infty}$ accounts for the total reward from time step $t = 0$ to infinity, with γ^t being the discount factor applied to future rewards R_{t+1} [31].

Action Value or Q value [33] gives the expected cumulative reward for an agent at state s taking an action a and following a policy π thereafter. This helps an agent in determining the favorability of choosing actions from that state by comparing the Q values associated with different actions [31]. The action value is mathematically defined as

$$Q_\pi(S_t, A_t) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_t, A_t \right] \tag{2}$$

where $Q_\pi(S_t, A_t)$ denotes the action-value function, which represents the expected cumulative reward when starting in state S_t, taking action A_t, and thereafter following policy π. The term $\mathbb{E}_\pi$ denotes the expectation over the sequence of actions and states generated by following policy π. The summation $\sum_{t=0}^{\infty}$ represents the total reward from time step $t = 0$ to infinity, with γ^t being the discount factor that discounts future rewards [31].

Categories of RL algorithms

RL algorithms can be categorized as the following:

On Policy and Off Policy RL algorithms [33, 31] The types of RL algorithms where the agent's behavior policy (the policy being used to pick the actions from a given state) is the same as the target policy (the policy being optimized to move it iteratively closer to the optimum policy) is called an On Policy algorithm. Whereas, when the behavior and target policies are not the same but rather work together by using the data generated by the behavior policy in the form of episode action-reward sequences to improve the decision rule generated by the target policy, the algorithm is called an off-policy RL algorithm.

Model based and Model free RL [33, 31] In model-based RL, the agent has access to the environment's internal model, based on which it makes informed decisions using reward predictions for actions from a given state. In contrast, model-free RL algorithms cannot access such a model. Instead, they strictly rely on the agent's direct interactions with the environment to explore and exploit learned rewards, which they then use to find the optimal policy.[33]

2.4 RL environments used in this thesis

This thesis focuses on the two of the most popular RL benchmarks, the Atari [3] and the DeepMind Control [35] environments. I used the Farama foundation's Gymnasium [38] environment implementations in my experiments. The following is a brief overview of the two benchmarks.

2.4.1 Atari

Assault Breakout Demon Attack Ms Pacman Pong

Figure 2: Popular Atari games

Gymnasium [38] uses the Stella [36] emulator to simulate the Atari games for RL development via the Arcade Learning Environment (ALE) [3]. The Atari benchmark [3] originally developed as console games [2] for human play is a very popular discrete action space benchmark used in testing and training RL algorithms. The full Atari 2600 benchmark consists of 57 games. However, because of the high computational requirements to solve all games, a smaller subset of 26 games trained until 100 thousand training steps is a popular alternative more commonly known as the Atari100k benchmark [16]. In this thesis, I focused specifically on the Atari100k to have faster self training results. These games consists of a wide variety of objectives with varying visuals and difficulty. These games have different reward densities, also some of these are hard exploration games. Therefore, offering a diverse set of environments for RL training. The reward structure of the benchmark differs in each game where often the player is required to dodge enemy attacks, collect points for items and duration survived in each gameplay. Some of the best selling games from this benchmark are Ms. Pacman, Pong, Breakout, Assault, Demon Attack etc.

The following table [7] lists the full set of actions from the discrete action space for all

games of this benchmark. Every game uses a subset of these actions based on the nature of the game.

Value	**Meaning**	**Value**	**Meaning**
0	NOOP	1	FIRE
2	UP	3	RIGHT
4	LEFT	5	DOWN
6	UPRIGHT	7	UPLEFT
8	DOWNRIGHT	9	DOWNLEFT
10	UPFIRE	11	RIGHTFIRE
12	LEFTFIRE	13	DOWNFIRE
14	UPRIGHTFIRE	15	UPLEFTFIRE
16	DOWNRIGHTFIRE	17	DOWNLEFTFIRE

Table 1: Action values and their meanings

Gymnasium offers rgb, grayscale and ram values as the observations for the Atari games. I used the default rgb images for training before applying the Atari preprocessing steps explained in [34] which most importantly convert the images to a square shaped grayscale, clips rewards between -1 and 1 along with some additional preprocessing steps.

2.4.2 DeepMind Control

Ball in Cup | Cartpole | Cheetah | Finger | Hopper

Figure 3: Popular DM Control environments

Just like Atari [3], I used Gymnasium's [38] integration for DeepMind Control Mujoco environments. Mujoco or Multi-Joint dynamics with Contact offers open source physics simulations for robotic tasks developed by Google's DeepMind [35]. These are the most popular continuous control environments with varying agent morphologies where the goal of the agent is to move and balance itself through a flat plane. Efficient movement and balancing gives the agent higher rewards depending on the type of the environment. Some of the popular DM Control environments are Ant, Cheetah run, Hopper, Pendulum, Cartpole, Ball in the cup etc. Some of these environments also offer sparse reward structures, for example the Cartpole balance/swingup sparse.

The continuous action space of these environments offer precise real values to control the torque and force applied to the agent's limbs and joints. Also, the state space of the DM Control environments consists of a combination of the position and velocity of the agent's body parts [38].

2.5 RL algorithms used in this thesis

The following subsections give a brief overview of the 6 RL algorithms used in the thesis. Namely the A2C [20], TRPO [28], PPO [30], Recurrent PPO [30], CURL [32] and the

Dreamer [12] algorithms.

2.5.1 A2C

Advantage Actor-Critic or A2C [20], is a versatile on-policy algorithm that can handle both discrete and continuous control environments. It uses two main components - the actor and the critic policies [10].

Actor - The actor policy provides a decision rule for the agent to select actions from different states.

Critic - Based on the actions taken by the actor, the critic policy calculates an advantage function to train the actor. The advantage function is defined in [20] as

$$A(S_t, A_t) = Q(S_t, A_t) - V(S_t) \tag{3}$$

where $A(S_t, A_t)$ represents the advantage function, which is the difference between the action-value function $Q(S_t, A_t)$ from equation 2 and the state-value function $V(S_t)$ from equation 1.

It takes a state-action pair and returns the difference between the observed return Q and the expected return V for an action from a state, quantifying the relative improvement of an action over the expected action value. If the advantage function comes out to be positive, the action is considered advantageous. Then the actor policy is updated to favor that action, thereby improving the actor policy.

2.5.2 TRPO

Trust Region Policy Optimization or TRPO [28], is an on-policy, policy gradient [33] algorithm compatible with both discrete and continuous action space environments.

As mentioned earlier in 2.3, an on-policy method is where the policy which generates roll out data (trajectories of state, action and rewards) is the same policy being updated and improved upon. This is done by moving the new policy in the direction of advantageous actions found by recognizing the positive returning advantage function actions. However, when applying policy gradient methods to an on-policy RL setting extra caution is needed because if the policy is updated too much from the original policy then this can lead to poor data collection in roll-outs from the new policy, leading to a downward spiral in the performance as the policies are updated based on the data they collect themselves [23, 41].

TRPO addresses this problem by applying a KL divergent constraint on the policy updates making sure that the policy updates are within the range of the so called Trust Region of the Trust Region Policy Optimization (TRPO) when new policy updates are accepted. The following are the brief mathematical equations and explanations for TRPO policy updates taken from [23].

$$\theta_{k+1} = \arg\max_{\theta} \mathbb{E}_{S,A\sim\pi_{\theta_k}} \left[L(S, A, \theta_k, \theta)\right] \quad \textbf{subject to} \quad \bar{D}_{KL}(\pi_{\theta_k}||\pi_\theta) \leq \delta \tag{4}$$

$$\textit{where } \bar{D}_{KL}(\pi_{\theta_k}||\pi_\theta) = \mathbb{E}_{S\sim\pi_{\theta_k}} \left[D_{KL}\left(\pi_{\theta_k}(\cdot|S)||\pi_\theta(\cdot|S)\right)\right] \tag{5}$$

$$L(S, A, \theta_k, \theta) = \rho(S, A, \theta, \theta_k) A(S, A) \tag{6}$$

$$\rho(S, A, \theta, \theta_k) = \frac{\pi_\theta(A|S)}{\pi_{\theta_k}(A|S)} \tag{7}$$

π_θ denotes a policy parameterized by θ, and θ_{k+1} represents the updated policy parameters obtained by maximizing the expected surrogate loss $L(S, A, \theta_k, \theta)$. The surrogate loss $L(S, A, \theta_k, \theta)$ uses the advantage function $A(S, A)$, scaled by the sampling ratio $\rho(S, A, \theta, \theta_k)$, which represents the ratio of the probability of action A under the new policy π_θ compared to the old policy π_{θ_k}. The constraint $\bar{D}_{KL}(\pi_{\theta_k}||\pi_\theta) \leq \delta$ ensures that the average Kullback-Leibler (KL) divergence between the old and new policies, $\bar{D}_{KL}(\pi_{\theta_k}||\pi_\theta)$, does not exceed the threshold δ, limiting the policy change in each update.

2.5.3 PPO

Proximal Policy Optimization or PPO [30], is an improvement and simplification of TRPO addressing the computational and implementation challenges associated with calculating the trust region, particularly the Kullback-Leibler divergence [15, 29]. This is implemented using a clipped surrogate objective updated via a mini-batch stochastic gradient descent. The following are the brief mathematical equations and explanations for PPO policy updates taken from [22].

$$\theta_{k+1} = \arg\max_\theta \mathbb{E}_{S,A\sim\pi_{\theta_k}} \left[L(S, A, \theta_k, \theta) \right] \tag{8}$$

$$L(S, A, \theta_k, \theta) = \min\left(\text{Ratio}, \text{Clipped Ratio}\right) \tag{9}$$

$$\text{Clipped Ratio} = \text{clip}\left(\rho(S, A, \theta, \theta_k), 1-\epsilon, 1+\epsilon\right) A^{\pi_{\theta_k}}(S, A) \tag{10}$$

$$\text{Ratio} = \rho(S, A, \theta, \theta_k)\, A_{\pi_{\theta_k}}(S, A) \tag{11}$$

where π_θ represents the policy parameterized by θ, and θ_{k+1} are the updated parameters obtained by maximizing the $L(S, A, \theta_k, \theta)$. $L(S, A, \theta_k, \theta)$ always chooses the minimum between the ratio of sampling ratio ρ, multiplied by the advantage function $A_{\pi_{\theta_k}}(S, A)$, and the clipped ratio. The clipped ratio introduces a constraint on the policy update by clipping $\rho(S, A, \theta, \theta_k)$ to the range $[1-\epsilon, 1+\epsilon]$, where ϵ is a clipping parameter that limits the change in the policy. This makes sure that the policy is not changed too much, improving the stability.

2.5.4 Recurrent PPO

Recurrent Proximal Policy Optimization or Recurrent PPO, is a version of PPO [30] designed for environments where an agent's decision depend on the past events or temporal dependencies for example in card games like poker where previous cards need to be remembered. This is done with the help of an RNN (Recurrent Neural Network) like LSTM (Long Short Term Memory) [39] enabling the agent to utilize information from previous time steps while updating the policy.

2.5.5 CURL

Contrastive Unsupervised Representations for Reinforcement Learning or CURL [32], is essentially a preprocessing step combined with standard model free RL algorithms where the raw image inputs of the environment are transformed to only contain the essential information from the states reducing the noise and irrelevant details.

CURL performs this with a constrastive loss which forces the states from the same action trajectories or states appearing close in time to be defined as positive samples. Similarly, it defines the states from different trajectories or distant in timesteps as negative samples. CURL learns these representations by training an encoder to map the positive samples closer to each other in the representation space. Finally, this encoder replaces the raw image input normally accepted by the model free RL algorithms with this transformed low dimension input for effective learning.

2.5.6 Dreamer

Dreamer [12] is an off policy model based RL algorithm which develops a compact and efficient world model in a latent space based on the interactions of an agent with the environment. Dreamer builds a recurrent state space using LSTMs [39] to utilize information from previous time steps to work efficiently with partially observable environments. Once this world model is created, it simulates trajectories of agent's transitions in the environment and optimizes the policy based on these simulations for a maximum cumulative reward.

Because of the world model and simulated trajectories, Dreamer does not need as many real environment steps than other RL algorithms, particularly in comparison to the model free methods [12].

2.6 Implementation Libraries

There are many popular implementation libraries for model free RL algorithms including Stable Baselines3 (SB3) [26] and ClearnRL [14]. I worked purely with Stable Baselines3 for self training RL agents as part of the experiments conducted in this thesis. Stable Baselines3 uses PyTorch [24] and abstracts the lower level implementation details of model free RL algorithms, providing an easy to use hyperparameter customisable API for fast prototyping and testing.

3 Related Work

Some recent research has been successful in reducing the computational costs of reinforcement learning (RL) benchmarks. The paper "Atari-5: Distilling the Arcade Learning Environment down to Five Games" [1] shows a subset of 5 representative Atari games as a mini Atari RL benchmark. This mini benchmark was able to predict the median scores of the full 57-game benchmark within 10% of true values at less than one-tenth of the computation costs. The paper used the Pearson Correlation Coefficient [4] to estimate similarities between many of the 57 games in the full benchmark, which they suggest made the compression possible. Similarly, the paper "Learning Representations for Pixel-Based Control: What Matters and Why?" [37] also notices a correlation between the performance of RL algorithms and categorized smaller benchmarks derived from a full benchmark. They suggest algorithms to be ranked based on data-centric properties instead of full benchmarks.

In terms of generalization, the paper "A Survey of Generalization in Deep Reinforcement Learning" [17] highlights the struggle of RL algorithms when faced with new unseen environments. They suggest that over-fitting in training environments/benchmarks is one of the reasons for the generalization gap in the test environments, underlying the importance of multi benchmark testing.

Taken together, these papers [1, 37, 17] are to the best of my knowledge, the most relevant papers related to my research. They suggest a need for further research associating RL algorithms' performance to environment properties and inspiring research to smartly choosing subset environments for feasible multi benchmark testing.

4 Concept

The paper "Atari-5" [1] introduces a compact set of five Atari games, which accurately approximates median scores of the full 57-game benchmark within 10% accuracy, using significantly less computational resources. Building on this foundation, my research aimed to extend its scope by identifying a representative subset not only from a single RL benchmark but from multiple benchmarks employing linear regression modeling. Based on this, the research question for my thesis was defined as the following.

Can a representative subset of environments across multiple RL benchmarks be found to reliably predict the overall benchmark performance?

I started with the goal of working with many different benchmarks like Atari [3], DM Control [35], Minihack [27], toy text and other classic control environments but because of the challenges encountered in building a usable algorithm score dataset, I limited my focus specifically to the Atari100k [16] and DM Control [35] environments. Also, these are two of the most popular and cited RL benchmarks, with the most amount of RL algorithm scores available.

4.1 Original planned approach

The original planned approach can be briefly explained in the following three main steps.

The first step was to gather algorithm scores from as many different environments as possible from benchmarks like Atari, DM Control, Minigrid, Classic Control, Toy Text, etc.
Secondly, the plan was to normalize all gathered scores using the random agent's score

from each environment as the baseline, along with an appropriate reference score, to make different benchmark scores comparable to each another.

Finally, linear regressions were to be performed, inspired by the Atari-5 paper [1]. This involved using all subset environment scores and environment properties (such as discrete/continuous action spaces, sparse/non-sparse rewards, etc.) of a chosen subset size to predict a summary score (for example, algorithms weighted median score) of the combined benchmark. The aim was to identify the subset with the best performing regression model.

4.2 Challenges and Adjustments

During the implementation process of the above mentioned planned approach, I encountered the following four challenges. I modified the process accordingly to circumvent these issues as mentioned below.

Firstly, I discovered that different research papers trained environments at different training steps. For example, the EfficientZero paper [42] showed DM Control scores at 100 thousand steps whereas the Dreamer V3 paper [12] showed DM Control results at 1 million environment steps, making the two algorithms incomparable. Therefore, it made sense to only consider algorithm scores that used the same number of training steps for a given environment.

Secondly, most research papers did not show algorithm scores for full benchmarks but only for a popular subset of environments from that benchmark. For example, the EfficientZero paper [42] used the Cartpole Swingup, Reacher Easy and Ball in cup catch environments [35] to verify their results on the DM Control environments. They did not give a justification on why this particular subset was chosen as the representative subset for the benchmark.

Thirdly, I encountered a major challenge that the set of algorithms results available for

one benchmark were totally different and non overlapping with algorithm results available for another benchmark. This lead to missingness in the score dataset not just for random individual cells but for entire consecutive blocks in the dataset corresponding to a benchmark. This became especially a challenge with the normalization process as the scores were normalized and scaled such that 0 represented a random performance and 100 represented our reference performance. So the missingness of data could not be imputed with 0 values as that was the same as implying that our missing algorithm score performed the same as a random agent which was definitely not true. Also using other imputation methods like taking mean and median of other available scores was not possible as the mean and medians would have been taken from different independent benchmark which had no proven direct relation to the the scores of the missing benchmark. Therefore, I decided to only work with algorithms which had no missing scores for environments in our combined dataset. Unfortunately, this drastically reduced the number of environments and algorithms that could be used and created a need for self training algorithm agents in order to extrapolate and expand the dataset.

Finally, The regression was planned such that the algorithm scores from a chosen subset of games predict the the median algorithm scores from all games in the benchmark [1]. More precisely in regression terminology, the input feature matrix X was planned to be a table with all algorithms as rows and a subset of games as columns listing the game scores for that algorithm in their respective cells. The target y was the algorithm median score across all games. This linear regression was chosen to find the subset of games which were the most predictive of the algorithm summary scores. However, including game properties like action space types, reward density became a challenge as these game properties were features of the regression features themselves (the game scores), which complicated the model structure. So in the end, I opted to just work with the algorithm scores and exclude the environment properties to maintain the one-to-one mapping between input features and output in the regression.

5 Methodology

This section explains the implementation methodology that I used in my thesis experiment. In the following subsections I describe the dataset creation, preprocessing, score normalization, the subset search, parallelization of experiments and the hyperparameters used.

5.1 Dataset Creation and Preprocessing

I began with Papers with Code [40] to gather research papers that listed RL algorithm scores for the Atari100k [16] and DM Control [35] benchmarks. I filtered out algorithms to only keep those algorithms with scores available for both the Atari100k and DM Control environments. Additionally, I ensured that all chosen algorithms listed scores in at least 10 environments from both benchmarks.

To maintain consistency across the scores for a given environment, I made sure that all algorithm scores for that environment used models trained with the same number of training steps. Furthermore, I used the Stable Baselines3 [26] and Stable Baselines3 Contrib [8] libraries to find implemented algorithms compatible with both discrete action space (Atari100k) and continuous action space (DM Control) to self-train models (My PPO [30], A2C [20], TRPO [28], and Recurrent PPO [30]) to expand the score dataset. PPO, CURL [32], and Dreamer [12] scores were imported from the literature [12]. Each algorithm was trained over 4 environment seeds (111,222,333 and 444 respectively) and finally these scores were averaged to obtain the individual game scores for an algorithm.

From these steps of combining RL algorithm scores from the literature and self-training algorithm scores, I was able to compile an initial working dataset with 26 Atari100k envi-

ronments and 12 DM Control environments, with scores available across 7 RL algorithms listed in 11.2. In this dataset, Atari scores were trained for 100k steps, and DM Control scores were trained for 1 million steps.

5.2 Score Normalization

I normalized the dataset scores using a random agent's score as the baseline (score = 0) and a self-trained Proximal Policy Optimization [30] (PPO) agent nearing convergence as the reference (score = 100). From this step all scores with a negative value meant that agent performed worse than a random agent, and all scores with a value more than 100 meant the agent performed better than our PPO agent nearing the set convergence criteria.

$$\text{normalized\_score} = \max\left(0, \frac{\text{score} - \text{base}}{\text{reference} - \text{base}} \times 100\right) \tag{12}$$

where *score* is the original score, *base* is the random agent's score and *reference* is the PPO agent's score nearing convergence.

Random agent scores were calculated over 4 seeds (111,222, 333, 444) with 10 trials containing 100 episodes each.

PPO agent scores nearing convergence were also calculated over 4 seeds (111,222,333, 444) where I utilized the Stable Baselines 3's StopTrainingOnNoModelImprovement callback [9] to set the convergence criteria. Using this callback I trained the all models for at least 1M training steps and triggered a callback to stop further training if the score

stopped improving in the last 250k training steps thereby setting a convergence criteria. The training plots for all the environments from both Atari100k and DM Control environments can be found in section 11.1.

After this step, I clipped all scores which performed worse than random (negative scores) to a 0 value and performed a log transformation on all scores as applied in the paper Atari5 [1].

$$\phi(x) := \log_{10}(1 + \max(0, x)) \tag{13}$$

where x denotes the score before transformation and $\phi(x)$ denotes the new log transformed score.

This methodology was aimed at ensuring comparability across scores from different environments and generalize the idea of human-normalized scores to the heterogeneous benchmark domains.

5.3 Subset Search

Finding the most representative subset from my combined benchmark was a 3 step process which involved calculating the weighted median scores from all algorithms, finding all possible environment subsets of a chosen size from the benchmark, and creating linear regressions for each of those subsets. These steps are explained in more detail as following.

After the score normalization from the previous step, I performed a multicollinearity

test between different algorithm scores in my dataset using a correlation matrix (Table 11.2). All correlation coefficients in the correlation matrix were found to be < 0.80 as observed in the table 10 indicating no high correlation between algorithms scores. Then, I calculated the weighted medians of each algorithm across all games where Atari100k and DeepMind Control had equal contributions to the median scores, despite the difference in the number of games between the two benchmarks. Then, I created all possible subsets of size 3 from the combined benchmark. After which I extracted the subset score tables where a row represented one of the 7 algorithms from 2.5 and columns contained the game scores from the respective game subset. Using the subset table as the regression input matrix X as shown in Table 2, and the weighted algorithm medians as the target y as shown in Table 3, I ran the linear regression.

$$y = X\beta + \epsilon \tag{14}$$

where y is the vector of weighted medians of the algorithm scores, X the input matrix formed by the subset game scores, β is the vector of regression coefficients and ϵ is the error term.

	game_1	game_2	..	game_m
algo_1	..	..	..	..
algo_2	..	..	..	..
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
algo_n	..	..	..	..

Table 2: Feature matrix X structure with n algorithm scores in m games.

$$\text{algo median}_1$$

$$\text{algo median}_2$$

$$\vdots$$

$$\text{algo median}_n$$

Table 3: Target vector y structure with n weighted median algorithm scores of the full benchmark.

Parallel linear regressions were run for all possible subsets of size 3. Just like Atari-5 [1], regression intercepts were set to false [1] to have a 0 score from a random policy. The linear regression models were then sorted by a 7 fold cross-validation mean square error in ascending order and r-square in descending order in case of a clash. The model that ranked first using this criteria was chosen as the best-fit linear regression, and its subset of games was selected as the most representative for the benchmark. The model with the least cross-validation mean square error was selected to avoid choosing overfit models. From the best subset found of size 3, best smaller subsets of sizes 2 and then 1 were identified following the same linear regression methodology. This made sure that the smaller subsets games were true subsets of the bigger subsets.

5.4 Parallelization of Experiments

All the RL training experiments were conducted in parallel over two University of Vienna, Data Mining Group servers [21]. Each of which featured 2x Intel 4214R CPUs, 2x 12 Cores, 2.4 GHz, 640 GB RAM, 2x NVIDIA A100 / 40 GB GPUs respectively.

All in all 760 RL agents were trained as part of this thesis. Specifically, 38 RL environments (26 Atari100k + 12 DM Control) trained for 5 RL algorithms (A2C, PPO, TRPO, Recurrent-PPO and PPO to convergence for normalization) for 4 environment seeds (111,222,333 and 444) per algorithm.

On both the servers, up to 12 processes per GPU were ran in parallel where each process picked up a dictionary containing RL environment, environment starting seed, and algorithm information to process from a shared task queue containing all tasks of required trainings. Python's multiprocessing module [11] and specifically the starmap function were used for multiprocessing. Similarly, parallel linear regressions to search the most representative subset from all possible subsets as explained in 5.3 were performed using CPU multiprocessing in the servers.

5.5 Hyperparameters Used

The specific hyperparameters used for self training PPO, A2C, TRPO and Recurrent-PPO algorithms for the Atari100k and DM Control environments were taken from the RL Baselines3 Zoo [25] tuned hyperparameters lists. The specific hyperparameters can be found in the section 11.4.

6 Results

From the 38 total environments in the combined benchmark, 8436 unique subsets of size 3 existed. I ran 8436 parallel linear regression models, where a regression model was created for each subset. The cross validation mean squared error (cv_mse) was used as the primary measure to rank the favorability of these models. I observed strong results from low cross validation mean squared error of 0.0035 and a high r square of 0.99 for the best model of size 3 despite the very limited number of algorithm samples in the regressions. From this best subset of size 3, a smaller best subset of size 2 and subsequently from that of size 1 were also derived. This was done to make the model performances easily upgradable from smaller subsets. The best subset of size 3 and its subsequent mini subsets are shown in the Table 4 along with their results summary.

Subset Size	**Environments**	**CV MSE**	**R^2**	**Relative Error**
3	Ms Pacman Ball in cup catch Pendulum swingup	0.0035	0.99	6.59%
2	Ms Pacman Pendulum swingup	0.0351	0.93	30.99%
1	Pendulum swingup	0.1380	0.66	72.37%

Table 4: Summary of Best Subsets

As can be seen in table 4, the three environments Ms Pacman, Ball in cup catch, Pendulum swingup were able to closely estimate the median algorithm performances of the full benchmark of 38 environments. Their linear regression model achieved this with a 6.59% relative error. Training these 3 games requires only 7.9% computational costs in comparison to the full benchmark.

7 Evaluation

The results in section 6 show that Ms Pacman, Ball in cup catch and Pendulum swingup were good predictors of the algorithm median scores of the full benchmark. However, the performance of these games rapidly drop with increased relative error as the smaller true subsets are derived from the best big subset. This indicates that the observed games are not very good predictors of the full benchmark individually, but rather only as a group. The scalability of the smaller subset of size 1 to the subset of size 2 and then the best subset of size 3 is not as robust as observed in a similar observation in the paper Atari-5 [1]. However, it is not unexpected as my regressions only contained scores for 38 environments from 7 algorithms whereas Atari-5 [1] had 57 environments with 62 algorithms scores in their regression dataset.

This following subsections examine the individual abilities of different games to estimate median algorithm scores of the benchmark. Further, the predictive performance of our best chosen subset in predicting individual game scores is evaluated. Both of these evaluations are inspired by [1].

7.1 Individual games predicting Median scores

I conducted 38 parallel linear regressions, one for each individual game, where I used the algorithm scores from that game to predict the median algorithm scores of the full benchmark. The following Tables 5 and 6 depict the structures of the regression input matrix X and target vector y.

	individual game score
algo_1	..
algo_2	..
$\vdots$	
algo_n	..

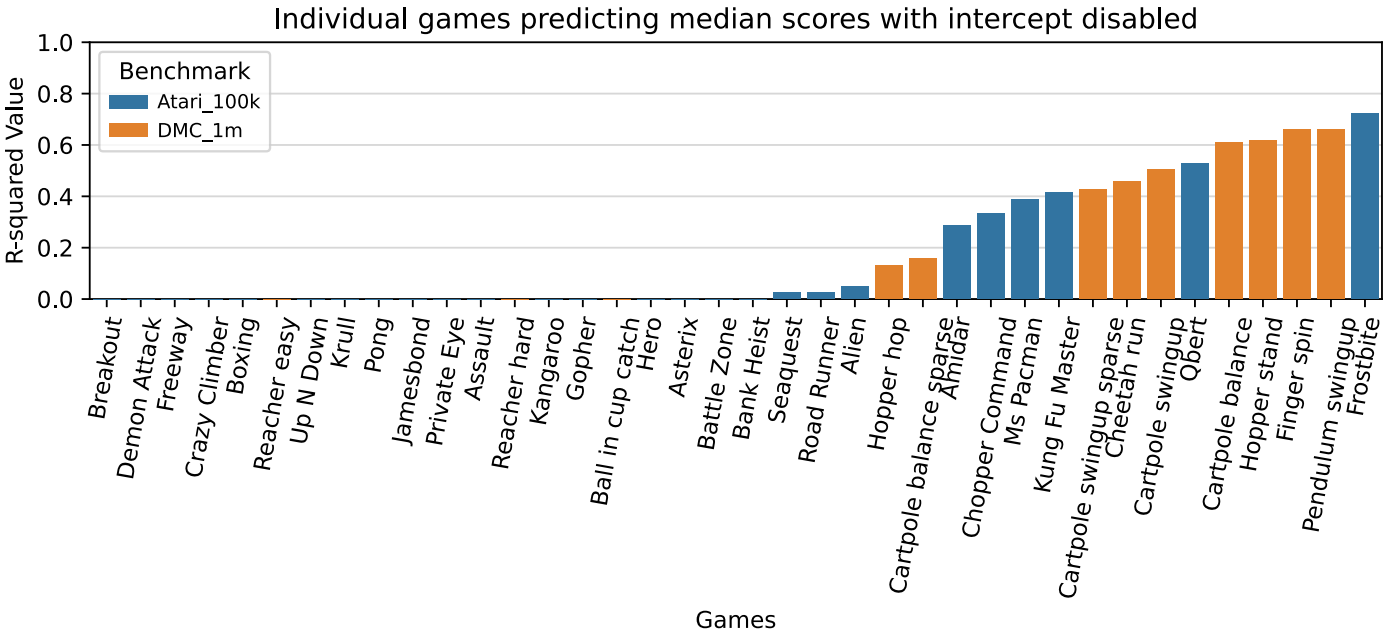
Table 5: Feature matrix X structure with n algorithm scores from an individual game.

$$\text{algo median}_1$$
$$\text{algo median}_2$$
$$\vdots$$
$$\text{algo median}_n$$

Table 6: Target vector y structure with n weighted median algorithm scores of the full benchmark.

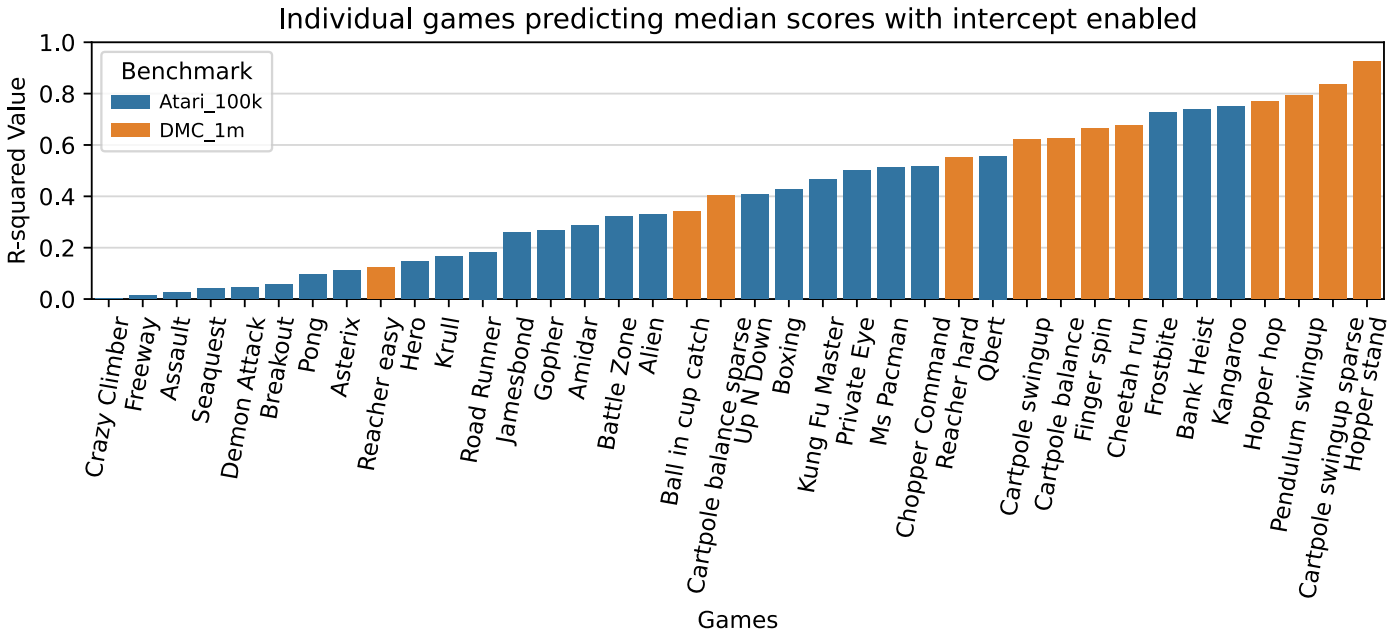
The following bar charts in Figures 4 & 5 plot the r-square values from individual games regression models estimating their ability to predict the overall benchmark's median algorithm scores with the intercept disabled and enabled respectively. In the cases if the r square values are negative, these values are capped at 0.

Figure 4



It was interesting to observe in Figure 4 that when the regression intercepts were disabled then many individual games returned a low or even negative r square value for their linear model. This showed that these games had low predictive ability towards the benchmark median algorithm scores when considered individually or their models or had an especially bad fit that failed to capture any data trends. However, when the same regressions were ran with the intercept enabled, then all r square values observed were positive. The full list of r square values is displayed in the Table 11 & 12.

Figure 5



We can see in Figure 5 that many games, particularly the atari games show low predictive ability when used by themselves. Hopper stand from DM Control had the highest individual performance with an r-square of 0.9262, whereas Crazy climber from Atari performed the worst with the lowest r-square of 0.0005.

7.2 Best Subset predicting individual game scores

For this evaluation experiment, I created 38 linear regression models where I predicted the algorithm scores for an individual game in each regression. The algorithm scores from the three game subset were used as the feature matrix for these regressions. Tables 7 and 8 depict the structures of the regression input matrix X and target vector y.

	game_1	game_2	game_2
algo_1	..	..	..
algo_2	..	..	..
$\vdots$	$\vdots$	$\vdots$	$\vdots$
algo_n	..	..	..

Table 7: Feature matrix X structure with n algorithm scores in games from the best subset of size 3.

algo_1
algo_2
$\vdots$
algo_n

Table 8: Target vector y structure with n algorithm scores of an individual game.

The bar charts in Figure 6 & 7 plot the predictive performance in r square values of Ms Pacman, Ball in cup catch and Pendulum swingup as a group in estimating the individual game scores from the full benchmark with the intercepts disabled and enabled.

Figure 6

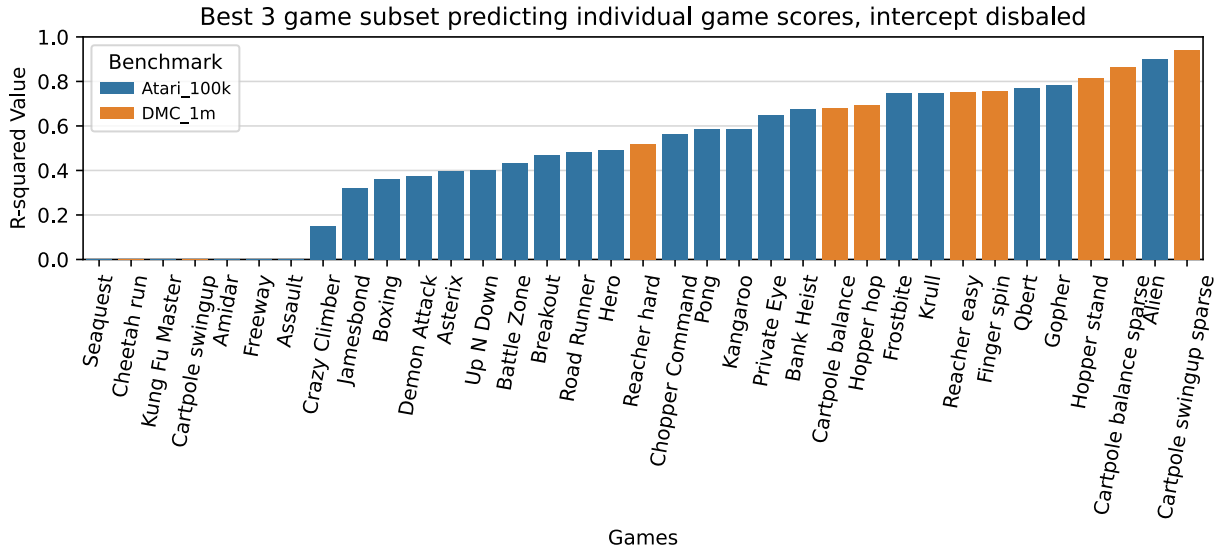
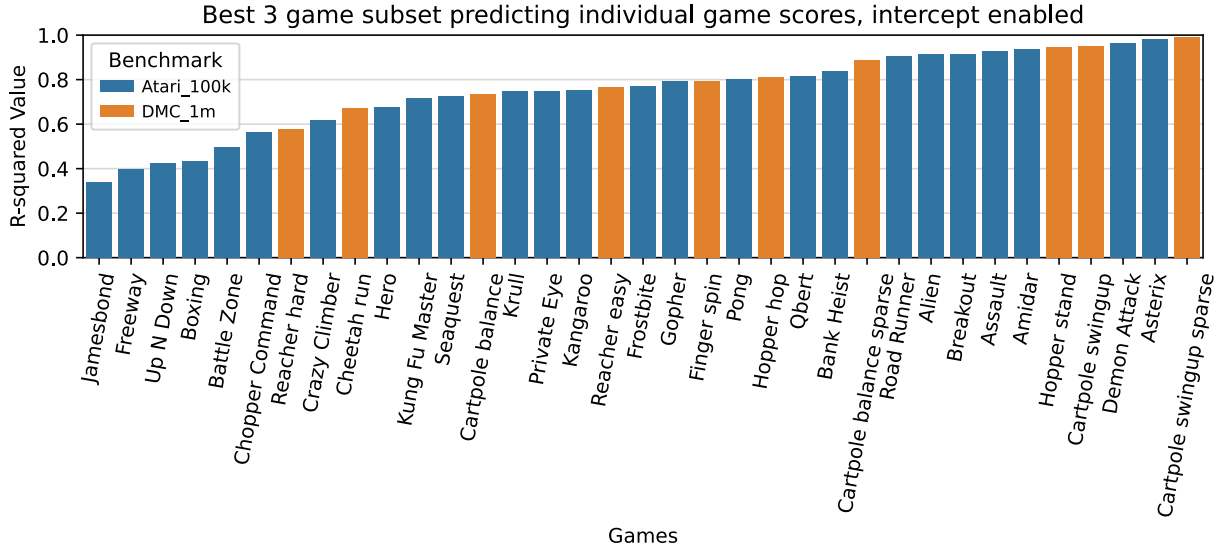


Figure 7



Just like Figure 4, when the intercepts were disabled then negative r square values capped at 0 were observed in Figure 6. Despite that I observed a robust performance with

high r-square values for most of the environments with both intercepts enabled and disabled. However, it was also noted that our best subset was still a poor fit for games like Jamesbond and Freeway with r-square values < 0.4 as can be seen in Figure 7. The full list of r square values is displayed in the Table 14 & 13.

7.3 Case Study

As part of this case study, I examined if the DreamerV3 paper [12] utilized the subset Ms Pacman, Ball in cup catch, Pendulum swingup, would the order of algorithm performance still be same as found in their original research.

They compared the performance of the DreamerV3 [12] algorithm with many other popular algorithms and discovered that the DreamerV3 scores outperformed CURL, which itself outperformed PPO in both the Atari100k and the DMControl1m benchmarks. The other algorithms mentioned in the paper are disregraded as they do not publish the results on all environments in our combined benchmark.

Therefore, I gathered the scores of Ms Pacman, Ball in cup catch, Pendulum swingup for the PPO, CURL and DreamerV3 algorithms as published in [12]. Since these three environments are from a combined benchmark of Atari100k and DMControl1m, normalization of scores was carried out in the same way as 5.2 before calculating the algorithm mean, median, weighted mean and weighted median performances. The algorithm summary scores of the subset are given in the Table 3 in the ascending order of the algorithm performances. It can be seen that the subset preserves the algorithm performance for the PPO, CURL and Dreamer algorithms in the same order as presented in the original paper.

Algorithm	Median	Weighted Median	Mean	Weighted Mean
PPO	0.55	0.55	0.91	0.82
CURL	1.75	1.70	1.90	1.85
DreamerV3	2.01	1.72	1.99	1.92

Table 9: Algorithm summary scores for the best 3 game subset

It should be noted that this case study's result may be biased. The reason is that DreamerV3 scores were included as a sample in the linear regression feature matrix used to identify the best subset, as explained in 5.1. I decided to conduct the case study on this paper because I could not find a new algorithm paper that published results for all of the subset environments. Also, I chose not to remove an existing algorithm from my training dataset for the case study, as that would reduce the number of samples in each regressions from 7 to 6 algorithms. When the number of samples is already so low, each sample is precious. Given the already limited number of available algorithms as samples, this reduction would have reduced the linear regression reliability and made it more prone to over fitting with an even smaller sample space. The number of folds in the cross validation analysis set to 7, would have been forced to decrement too. Hence, I decided to use all the 7 algorithm scores in the regression dataset and acknowledge the possibility of bias in this case study.

8 Discussion

This chapter discusses the limitations, ideas for future works and the overall impact of this research.

8.1 Limitations

Three main limitations from my experiments in this thesis can be listed below.

Firstly, the focus of my thesis was limited to all 26 Atari100k [16] and 12 DM control [35] environments because of the challenges encountered in building a big dataset with usable algorithm scores available in the literature as explained in detail in section 4.2. Having linear regression models with a handful of samples because of the less number of algorithm scores available was the biggest limitation of my research. Therefore, working with more algorithms and benchmarks for a bigger dataset would be the natural direct extension to this thesis if future researchers are able to counter the aforementioned challenges with score collection. Having more algorithm scores available for the environments will strengthen the reliability of results, as adding algorithms will add samples in the linear regression enhancing the robustness.

Secondly, while self-training environments, particularly with Atari environments, I noticed a high standard deviation in the algorithm scores when using different seeds. I used 4 unique seeds for calculating the mean scores. However, working with more environment seeds could be interesting and more reliable in the future experiments.

Thirdly and finally, the goal of my thesis experiment was to to find the most representative environments from two different benchmarks where the environments were of discrete action space in Atari and continuous action space in DM Control. However, this thesis can be extended to work with benchmarks of specific categories, such as working

with benchmarks where all environments are, for example, of discrete action space. This way, the number of overlapping algorithms between the benchmarks being studied will definitely go up.

8.2 Future Work

Building upon the limitations identified in the previous section, here I list four ideas for future work that emerge as potential extensions of this project.

Working with more algorithms and benchmarks for a bigger dataset would be a natural extension, assuming future researchers can address the challenges in score collection mentioned in the section 4.2. More algorithm scores will enhance the reliability of results by adding samples for linear regression.

Secondly, Extending the experiments to work with other singular categories, like continuous action space, sparse or dense reward environments, or hard exploration games, could allow researchers to identify the most important environments from specific singular categories.

Thirdly, My research only used algorithm scores as samples for regression due to the one-to-one mapping structure required between the input and output of regressions. Future researchers could focus on employing more complex models using neural networks, which add features for environment properties like action space types, observation space types, reward density, and game categories in addition to game scores. This could provide an explanation for the compression of results that the regression is able to recognize with the best subset found.

Finally, it would be interesting to find the most representative subset of environments from the same benchmark where each environment is tested on different training steps. For example, comparing Atari100k and Atari1m might lead to some interesting findings, showing some cheaply trained environments as more important than others with higher

training steps.

8.3 Impact

The impact of this thesis could be directly seen in terms of making RL algorithms more feasible to train and test with multiple RL benchmarks using a smartly chosen subset of environments instead of full benchmarks with the saved computation and time resources. Additionally, this proof of concept thesis could be extended to identify the most important RL environments from specific categories. For example, the most important RL environments that researchers should focus on when working with individual categories like discrete/continuous control environments or sparse/dense reward environments or hard exploration games respectively. The benefit of using multiple benchmarks could contribute to generalisability improvements of new RL algorithms [17].

Also, while working on this thesis I ran into the challenge of seeing a huge mismatch between benchmark popularities with many results published for a few popular benchmarks like Atari and DM Control and significantly less published results for other benchmarks like Procgen [6], Minihack [27] etc. Hopefully with training becoming more feasible, this thesis inspires researchers to publish algorithm performance results on more benchmarks. Finally, I noticed no standardization in the training steps and environment seeds being used when publishing results for specific RL environments. Different papers publish results using different settings. For example, some papers published the results on 100k steps of DM Control whereas others directly for 1M steps omitting scores for 100k steps making the two research papers difficult to compare. Although, we have seen some progress in this regard with the popularity of benchmarks like Atari100k but this should be extended to other benchmarks too as that would make research like mine easier to conduct in the future.

9 Conclusion

This proof of concept research provides a methodology as an extension to the paper Atari-5 [1] to find the the most representative subset of environments from multiple RL benchmarks for efficient multi-benchmark testing. This thesis introduces a technique for normalizing algorithm scores across multiple RL benchmarks, enabling environment scores from different benchmarks to be comparable to one another. The normalization process utilizes the performance of a random agent as a baseline and an appropriate reference, such as a PPO agent's score at convergence. The scores are then log-transformed to facilitate comparability between different environments. Following this normalization, the methodology introduced by the Atari-5 paper [1] is applied to create linear regression models on benchmark subset scores to predict full benchmark summary scores, such as the weighted median score from all algorithms. This approach identifies the environments that form the most representative subset for the combined benchmark based on the best performing regression model derived from all subset models.

I applied this methodology to the Atari100k and a subset of DMControl1m benchmarks. The three environments Ms. Pacman, Ball in Cup Catch, and Pendulum Swingup, identified through linear regressions from all possible subsets of size 3 were able to predict the algorithm summary (median) scores with a relative error of 6.59% compared to the full benchmark. Training on these three games, rather than the full set of 38 games, requires only 7.9% of the computational costs, leading to significant computational savings.

This thesis will hopefully inspire an RL research expanding the scope of my project to an even bigger pool of diverse benchmarks. Also, this hopefully leads to a less arbitrary choice of environment subsets selection when developing new RL algorithms that reap the benefits of multiple benchmarks testing in a feasible way.

10 References

[1] Matthew Aitchison, Penny Sweetser, and Marcus Hutter. Atari-5: Distilling the arcade learning environment down to five games, 2022.

[2] Atari. Atari 2600 plus. https://atari.com/products/atari-2600-plus?srsltid=AfmBOooQlZYDukIXvAr-IL6bwfObEVe0jKquWoI0dnYrZSiUO3qdfnnU, 2024.

[3] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, June 2013.

[4] J. Benesty, J. Chen, Y. Huang, and I. Cohen. Pearson correlation coefficient. In *Noise Reduction in Speech Processing*, volume 2 of *Springer Topics in Signal Processing*, pages 43–58. Springer, Berlin, Heidelberg, 2009.

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[6] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning, 2020.

[7] Gym Developers. Gym atari environments. https://www.gymlibrary.dev/environments/atari/index.html#action-space, 2024.

[8] SB3-Contrib Developers. Sb3-contrib documentation. https://sb3-contrib.readthedocs.io/en/master/, 2024.

[9] Stable-Baselines3 Developers. Stable baselines3: Stoptrainingonnomodelimprovement callback. https://stable-baselines3.readthedocs.io/en/master/guide/callbacks.html#stoptrainingonnomodelimprovement, 2024.

[10] Hugging Face. Deep reinforcement learning: Advantage actor critic (a2c). `https://huggingface.co/blog/deep-rl-a2c`, 2023.

[11] Python Software Foundation. multiprocessing — process-based parallelism. `https://docs.python.org/3/library/multiprocessing.html`, 2024.

[12] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models, 2024.

[13] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters, 2019.

[14] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

[15] James M. Joyce. *Kullback-Leibler Divergence*, pages 720–722. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[16] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model-based reinforcement learning for atari, 2024.

[17] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of zero-shot generalisation in deep reinforcement learning, 2023.

[18] Timo Klein. Reinforcement learning meets transfer learning - topic proposal slides. Presentation Slides, 2023. Contact the author for access.

[19] Heinrich Küttler, Nantas Nardelli, Alexander H. Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment, 2020.

[20] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

[21] University of Vienna. Server hardware - getting started. https://wiki.univie.ac.at/pages/viewpage.action?spaceKey=DataMining&title=Server+Hardware+-+Getting+Started, 2024.

[22] OpenAI. Proximal policy optimization (ppo). https://spinningup.openai.com/en/latest/algorithms/ppo.html, 2024.

[23] OpenAI. Trust region policy optimization (trpo). https://spinningup.openai.com/en/latest/algorithms/trpo.html, 2024.

[24] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[25] Antonin Raffin. Rl baselines3 zoo. https://github.com/DLR-RM/rl-baselines3-zoo, 2020.

[26] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[27] Mikayel Samvelyan, Robert Kirk, Vitaly Kurin, Jack Parker-Holder, Minqi Jiang, Eric Hambro, Fabio Petroni, Heinrich Küttler, Edward Grefenstette, and Tim Rocktäschel. Minihack the planet: A sandbox for open-ended reinforcement learning research, 2021.

[28] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017.

[29] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017.

[30] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[31] David Silver. Lectures on reinforcement learning. URL: `https://www.davidsilver.uk/teaching/`, 2015.

[32] Aravind Srinivas, Michael Laskin, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning, 2020.

[33] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[34] Daniel Takeshi. Frame skipping and preprocessing for deep q-networks on atari 2600 games, 2016.

[35] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. Deepmind control suite, 2018.

[36] Stella Development Team. Stella - atari 2600 emulator. `https://github.com/stella-emu/stella`, 2024.

[37] Manan Tomar, Utkarsh A. Mishra, Amy Zhang, and Matthew E. Taylor. Learning representations for pixel-based control: What matters and why?, 2021.

[38] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023.

[39] Christian Bakke Vennerød, Adrian Kjærran, and Erling Stray Bugge. Long short-term memory rnn, 2021.

[40] Papers with Code. Papers with code. `https://paperswithcode.com/`, 2024.

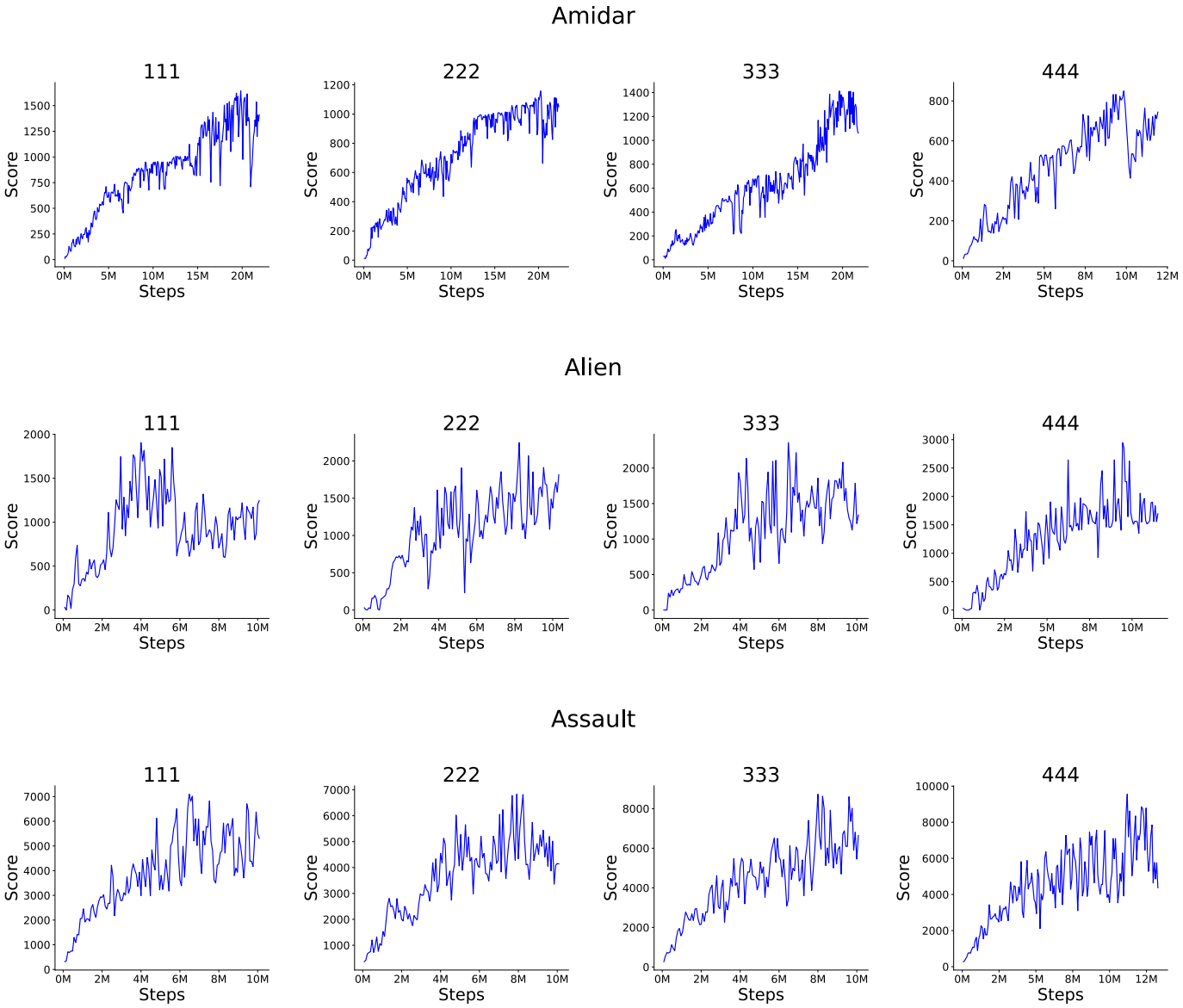
[41] Papers with Code. Trust region policy optimization (trpo). `https://paperswithcode.com/method/trpo`, 2024.

[42] Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering atari games with limited data, 2021.

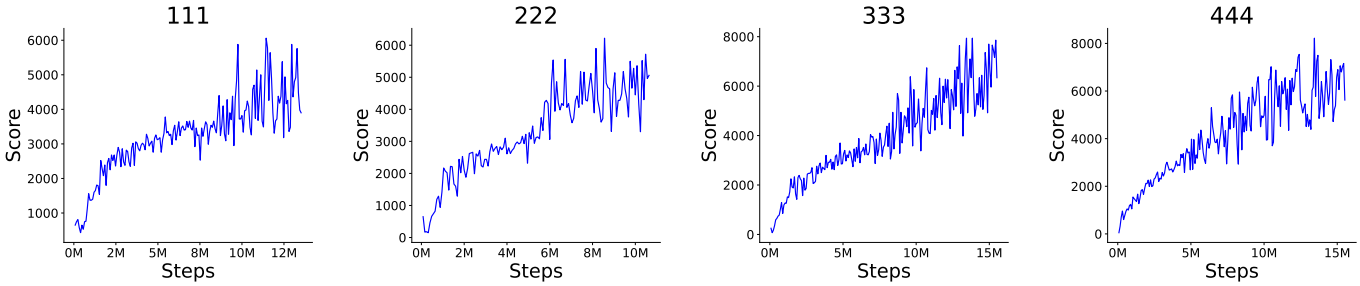
11 Appendix

11.1 PPO Convergence plots

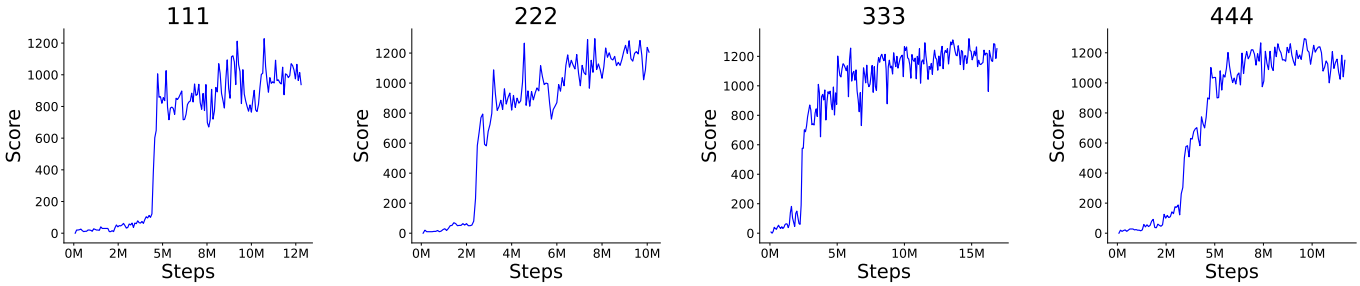
Atari



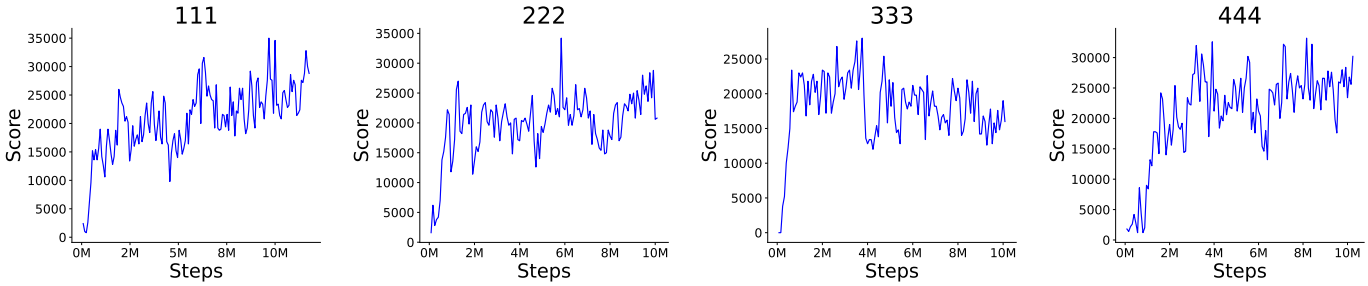
Asterix



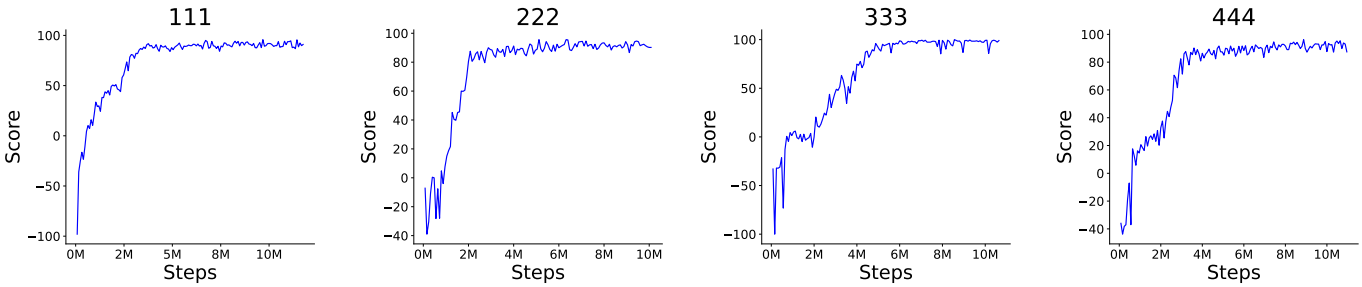
Bank Heist



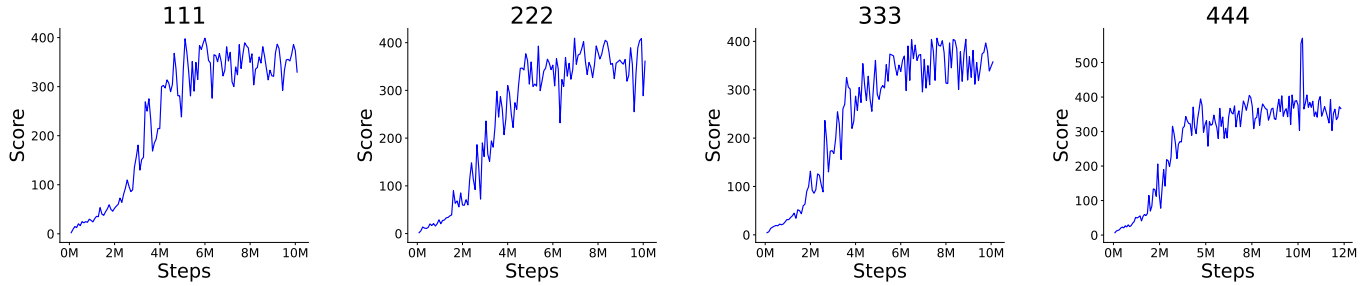
Battle Zone



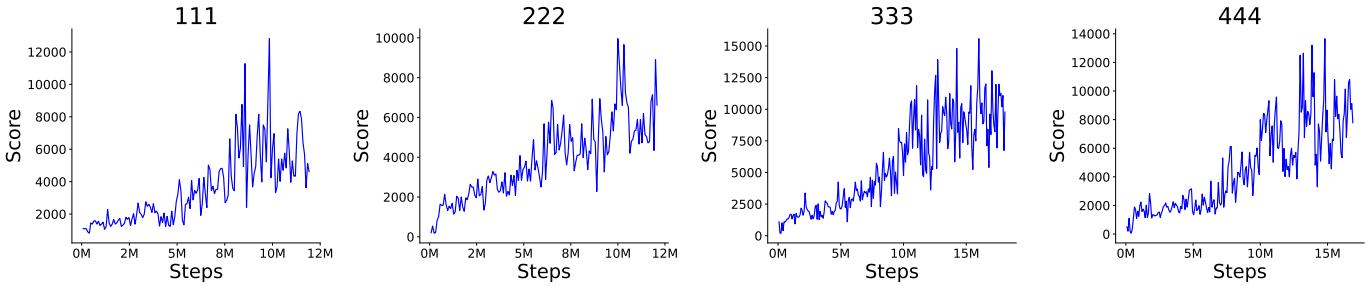
Boxing



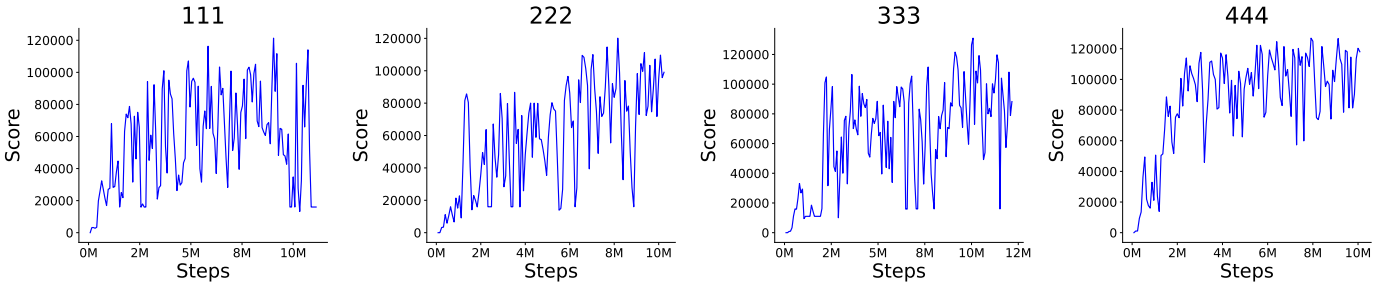
Breakout



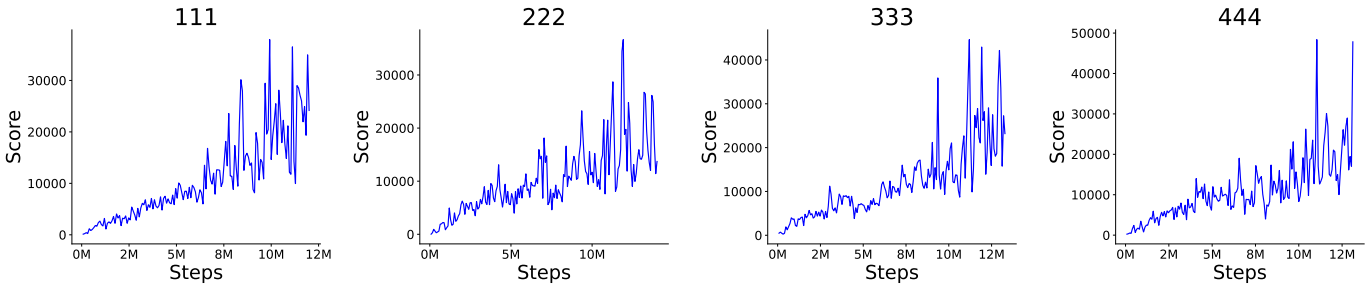
Chopper Command



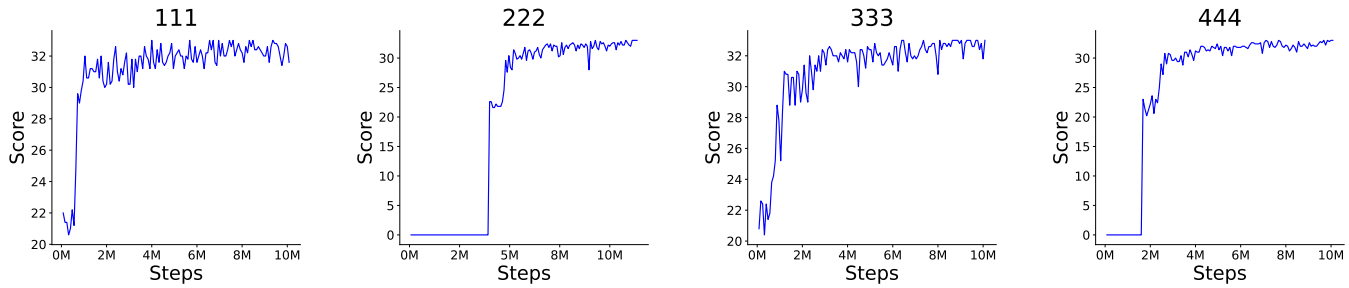
Crazy Climber



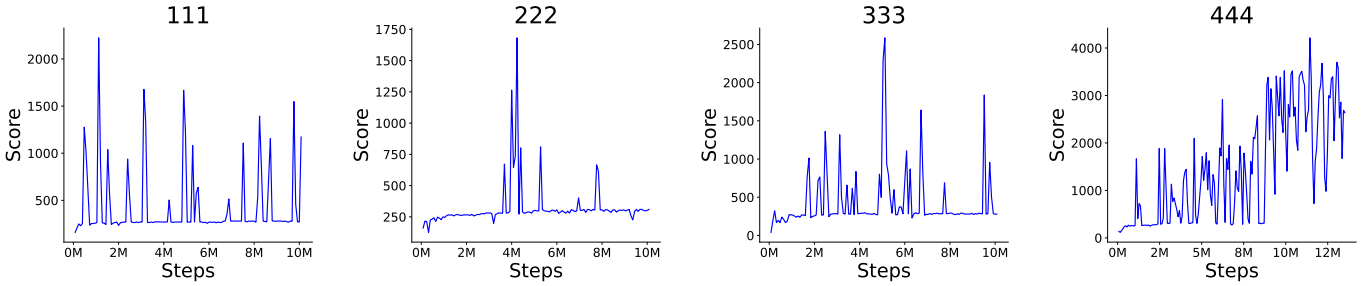
Demon Attack



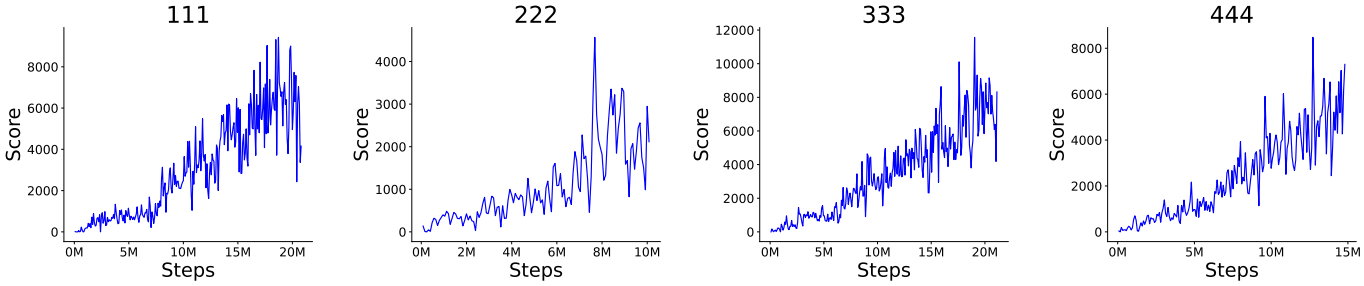
Freeway



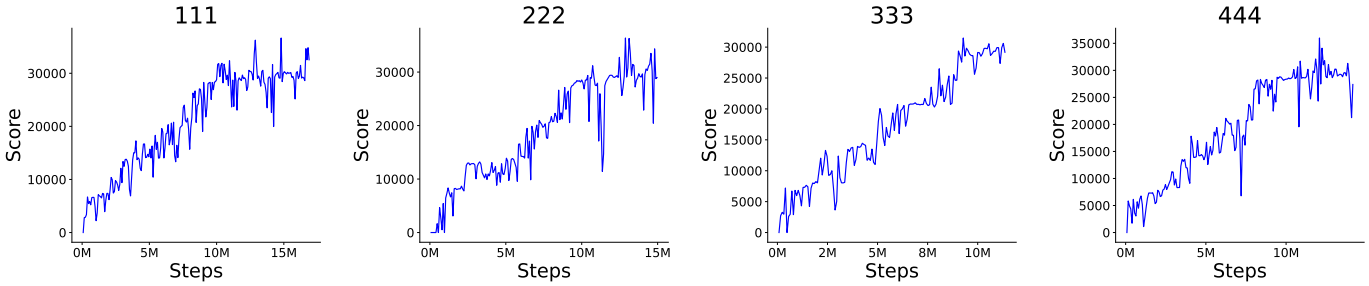
Frostbite



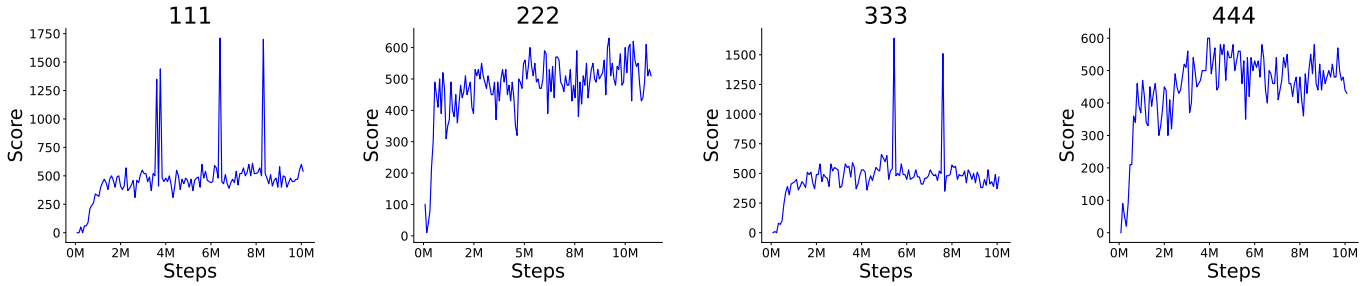
Gopher



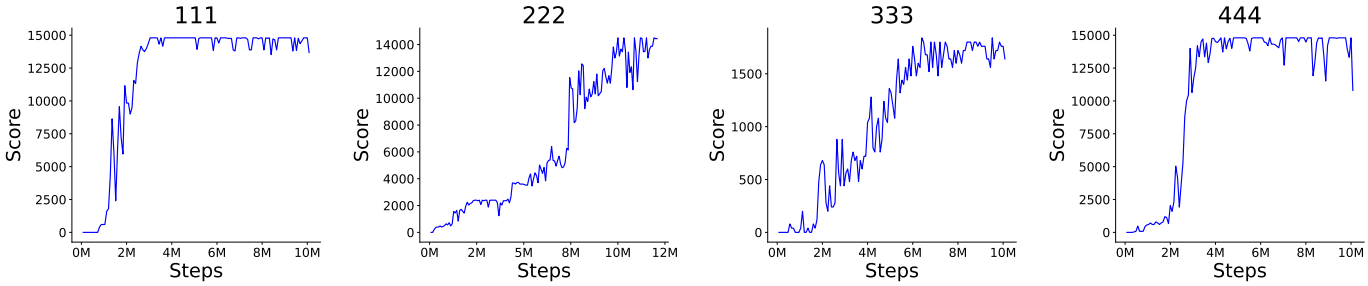
Hero



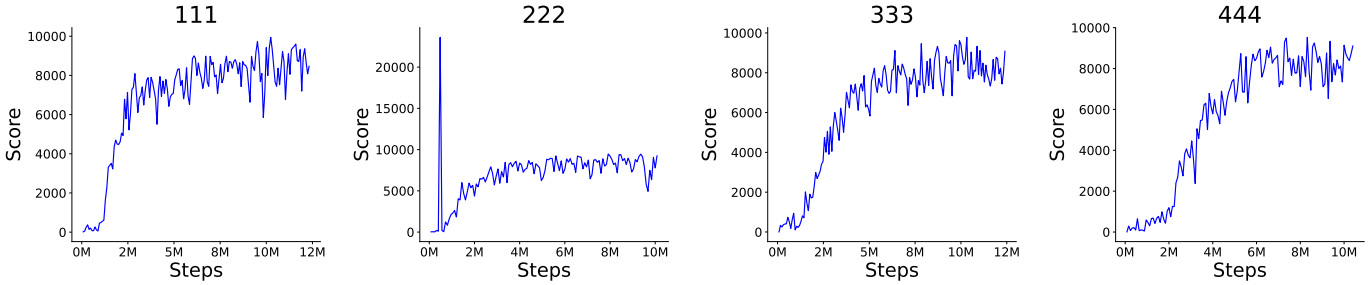
Jamesbond



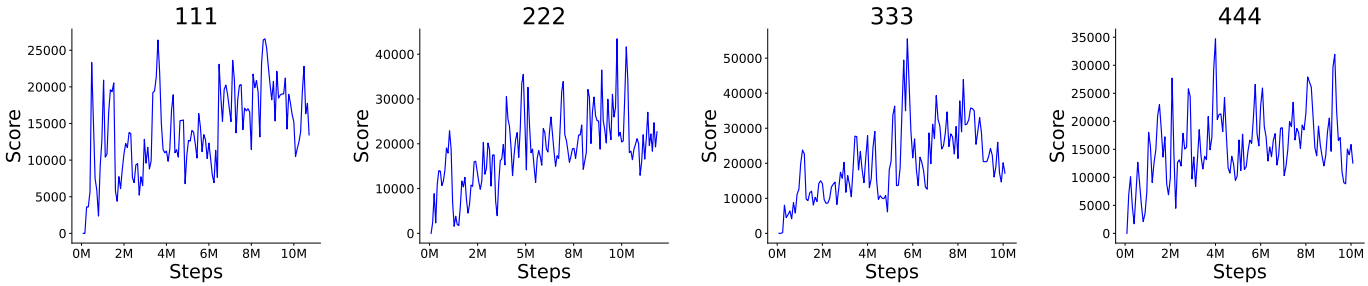
Kangaroo



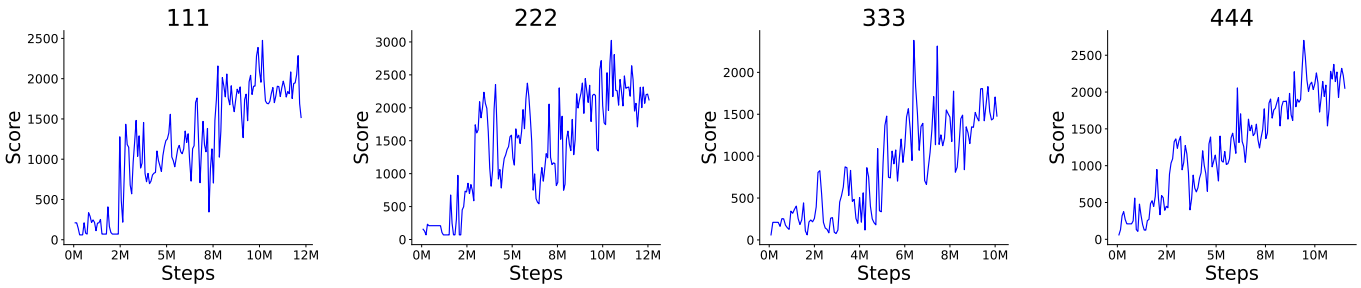
Krull



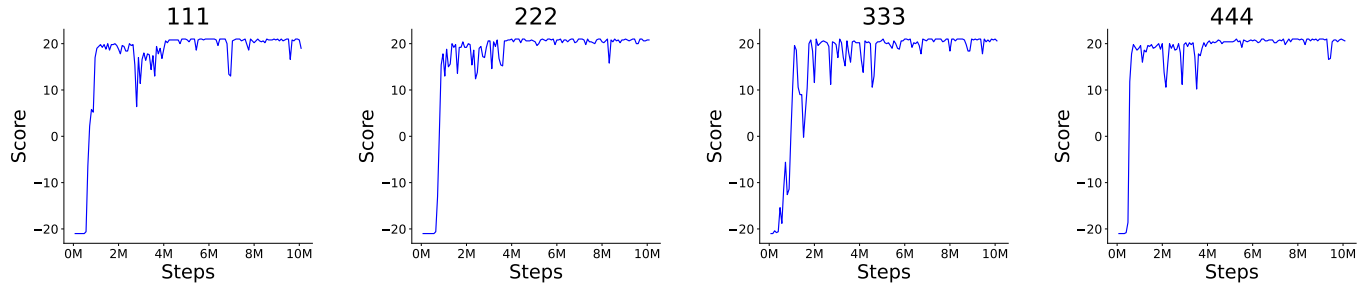
Kungfu Master



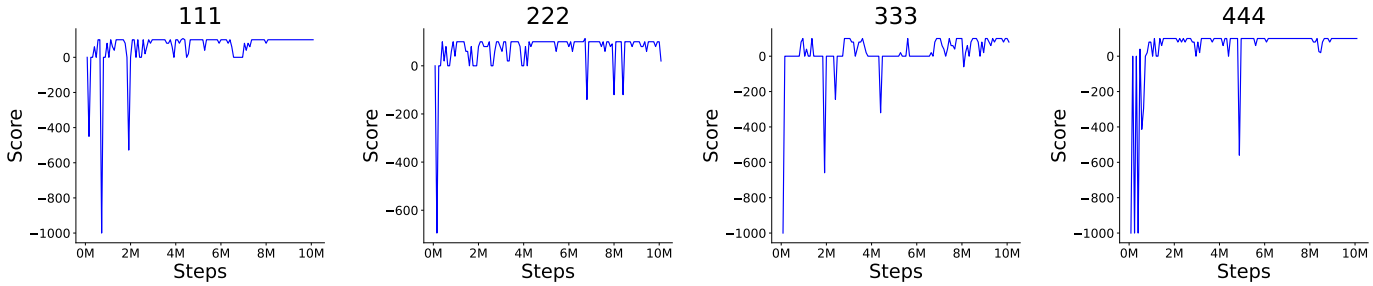
Ms Pacman



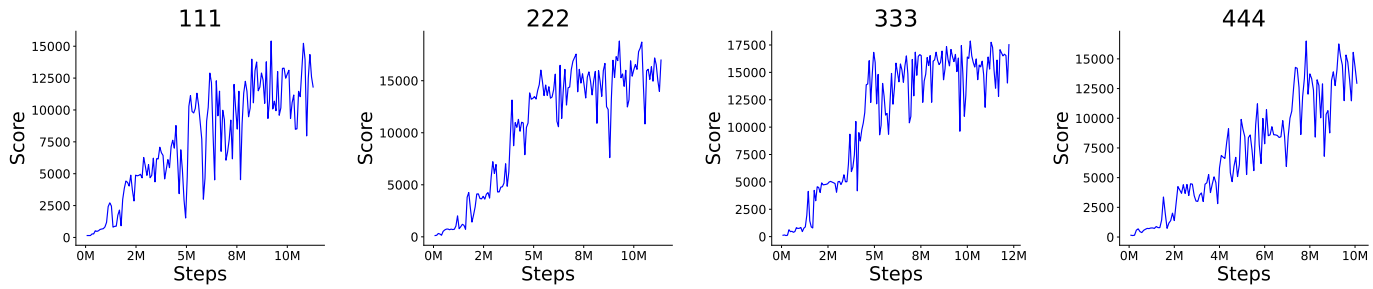
Pong



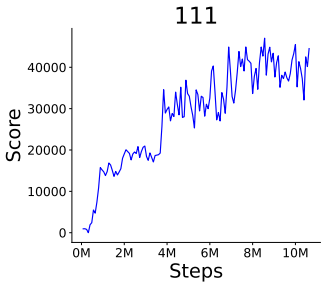
Private Eye

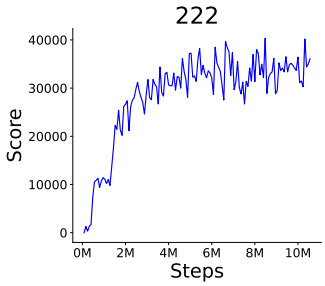


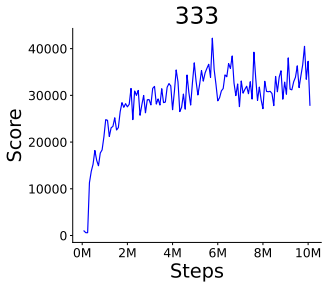
Qbert

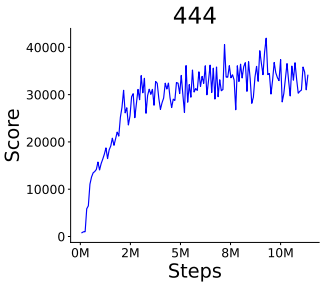


Road Runner

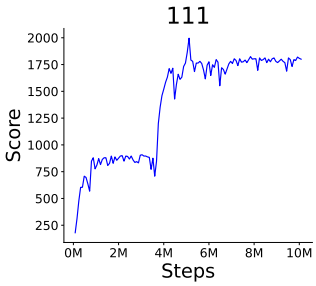


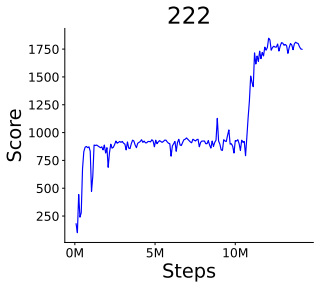


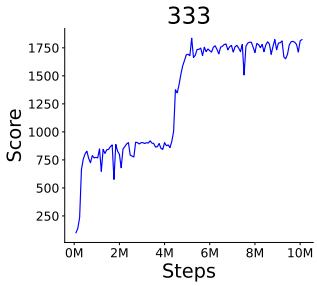


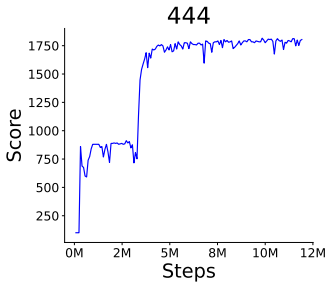


Seaquest

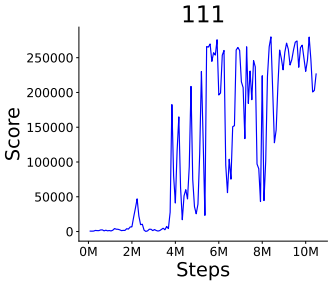


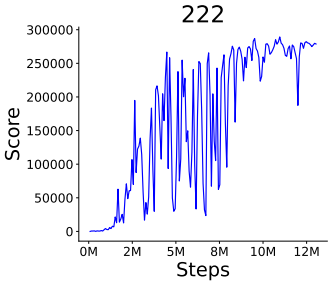


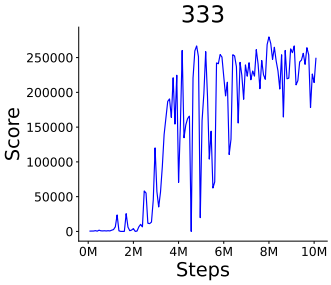


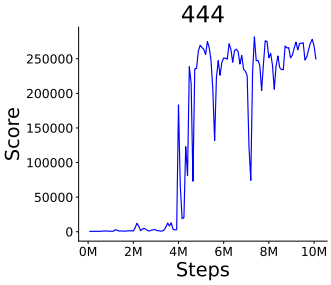


Up N Down



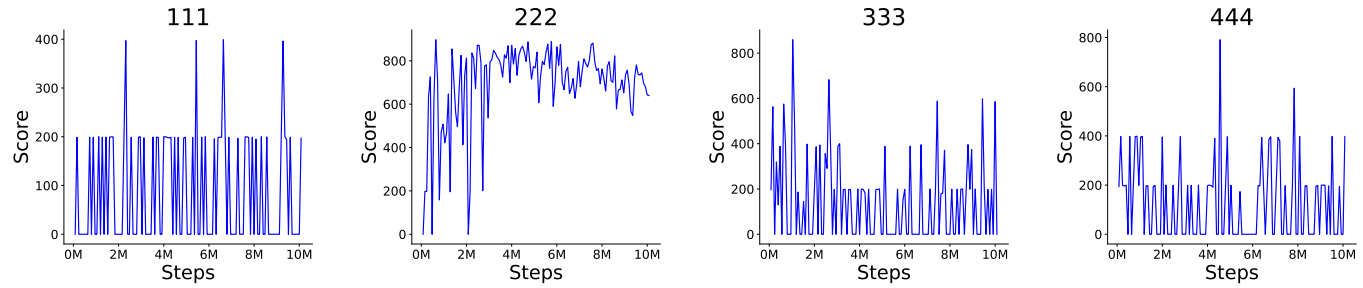




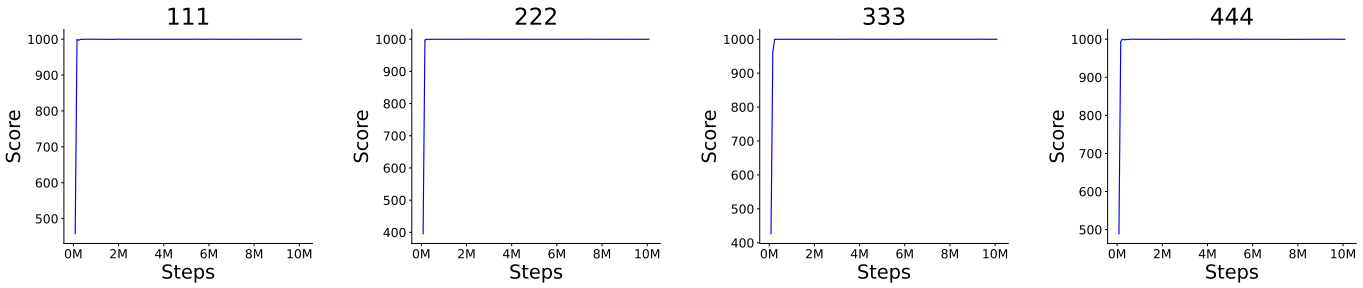


Deep Mind Control

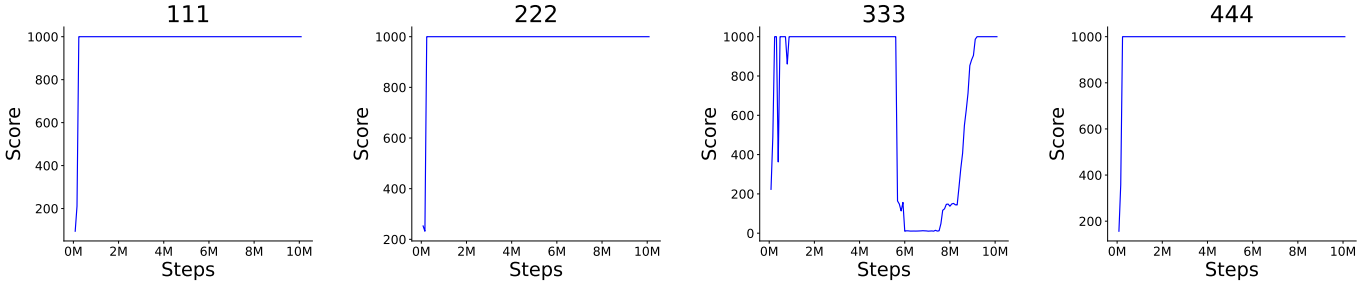
Ball in cup catch



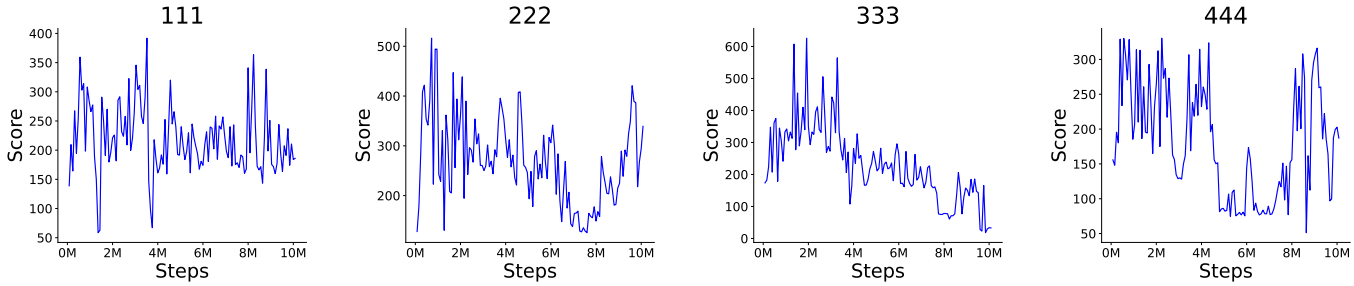
Cartpole balance



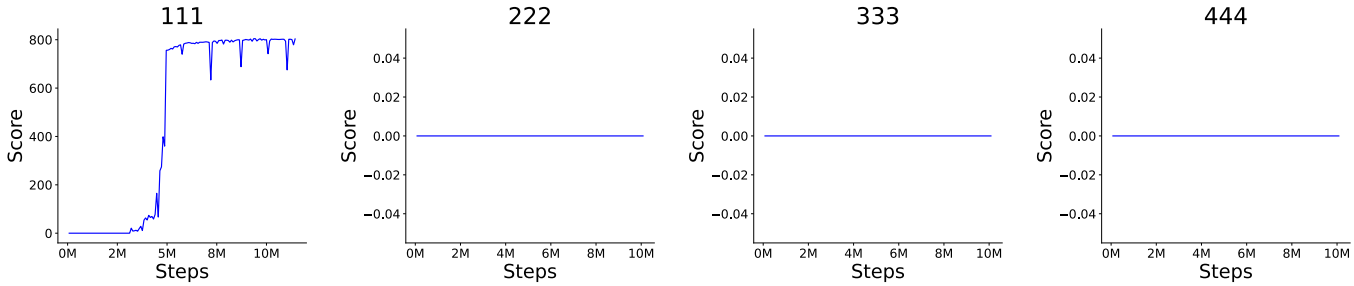
Cartpole balance sparse



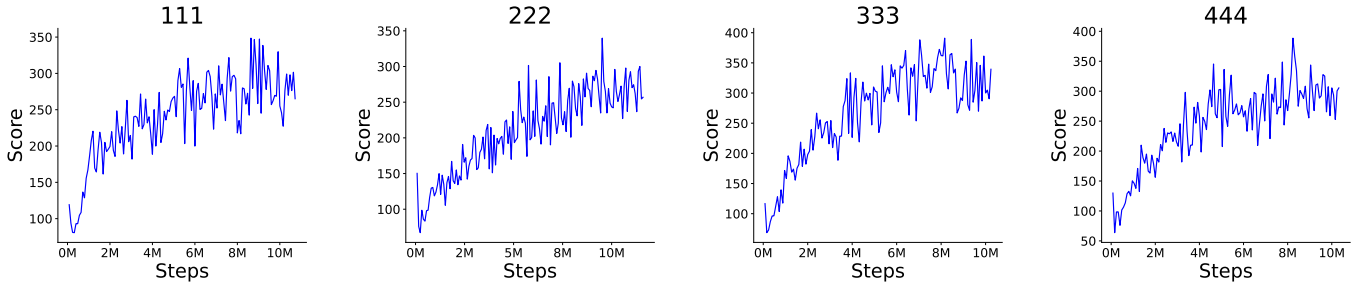
Cartpole swingup



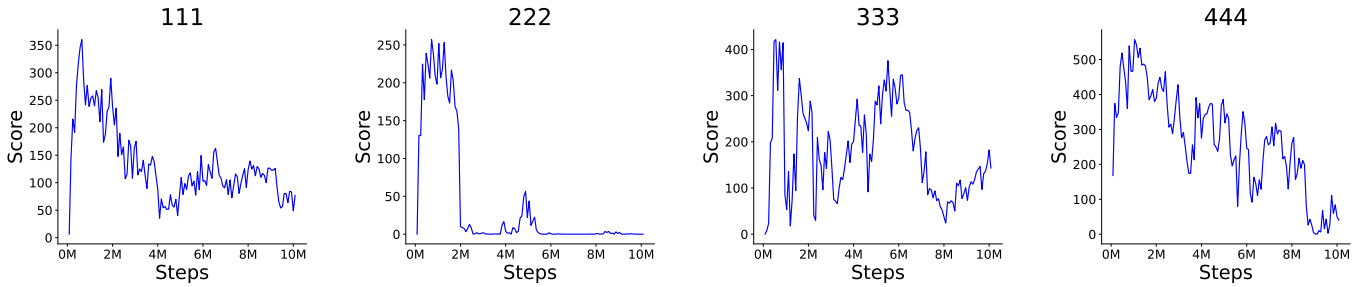
Cartpole swingup sparse



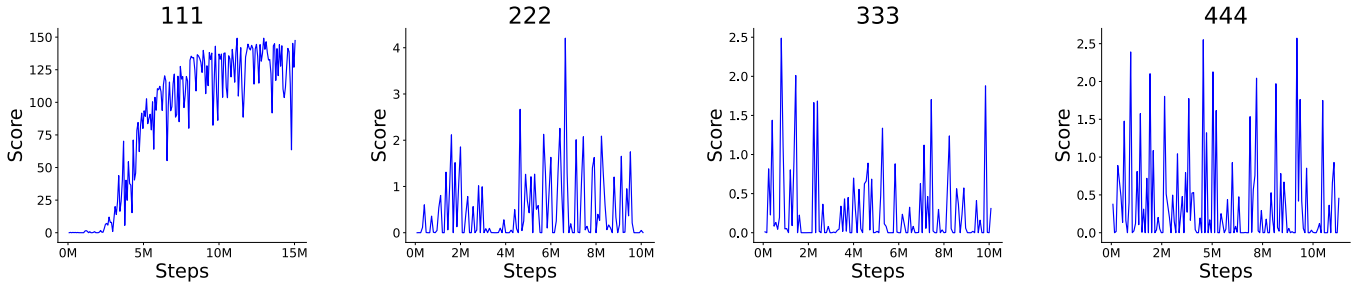
Cheetah run



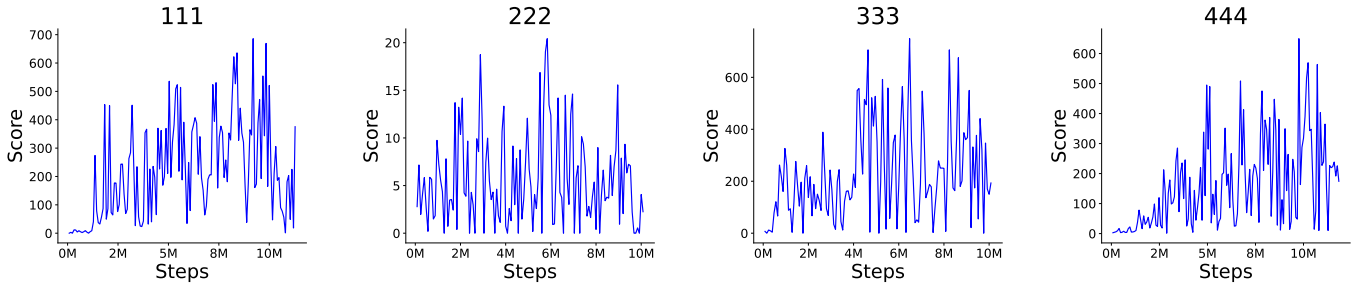
Finger spin



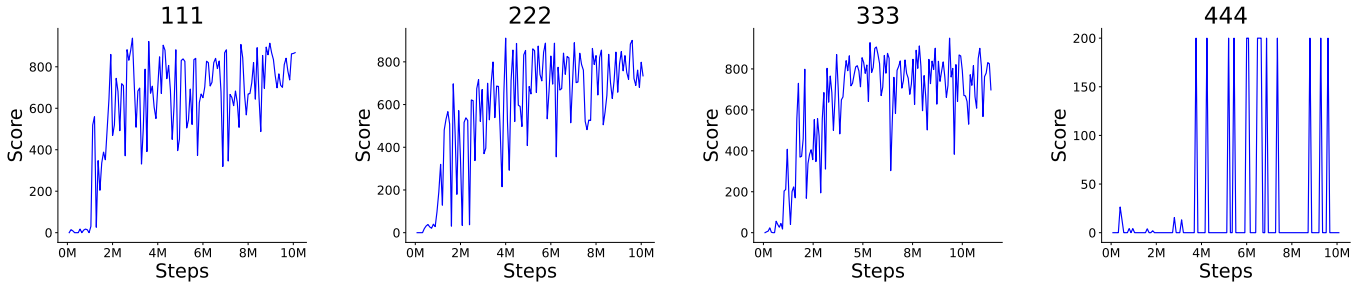
Hopper hop



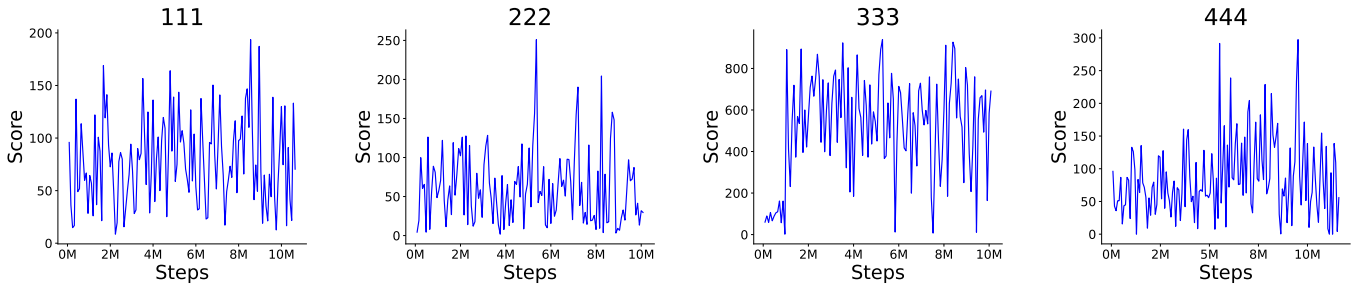
Hopper stand

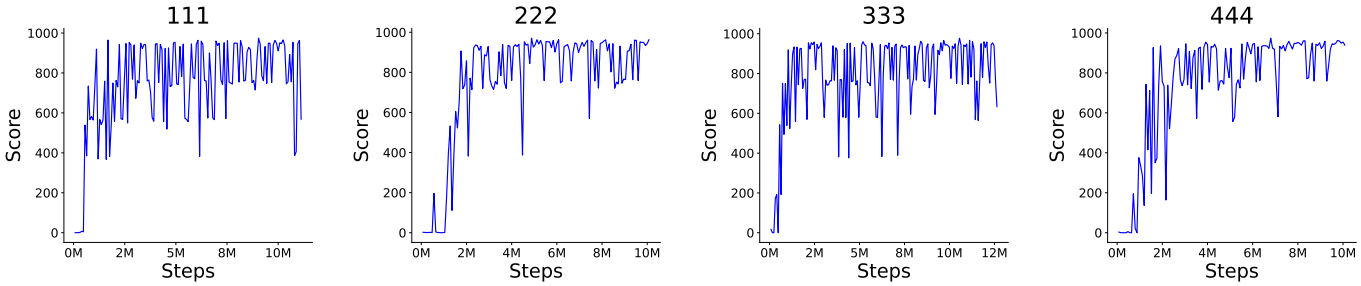


Pendulum swingup



Reacher easy




Reacher hard
111
222
333
444
Score
Steps
0
200
400
600
800
1000
0M
2M
4M
5M
6M
8M
10M
12M

11.2 Game Scores

Environments	Random	Reference	My PPO	A2C	TRPO	R-PPO	PPO	CURL	Dreamer
Alien	227.80	2364.50	413.25	615.00	179.75	301.50	276	558.20	1118
Amidar	5.80	1266.00	129.08	136.30	54.35	74.12	26	142.10	97
Assault	222.40	8051.60	1267.90	1571.40	352.80	670.42	327	600.60	683
Asterix	210.00	7110.00	1090.00	1911.25	338.75	681.25	292	734.50	1062
Bank Heist	14.20	1284.50	44.75	27.00	13.75	22.00	14	131.60	398
Battle Zone	2360.00	32600.00	17075.00	2625.00	3575.00	14050.00	2233	14870.00	20300
Boxing	0.10	96.90	4.60	-29.88	-27.35	-23.70	3	1.20	82
Breakout	1.70	446.35	25.10	209.32	1.92	26.92	3	4.90	10
Chopper Command	811.00	13005.00	1372.50	1060.00	867.50	885.00	1005	1058.50	2222
Crazy Climber	10780.50	124845.00	47247.50	91152.50	4745.00	53312.50	14675	12146.50	86225
Demon Attack	152.10	41902.50	1177.25	5013.38	338.00	2090.75	160	817.60	577
Freeway	0.00	33.00	22.32	0.00	20.92	22.85	2	26.70	0
Frostbite	65.20	2676.50	250.75	278.25	261.50	252.75	127	1181.30	3377
Gopher	257.60	8510.00	442.00	832.00	147.50	486.00	368	669.30	2160
Hero	1027.00	35119.00	5002.88	4729.00	503.00	4997.12	2596	6279.30	13354
Jamesbond	29.00	1145.00	477.50	33.75	8.75	300.00	41	471.00	540
Kangaroo	52.00	11485.00	500.00	30.00	15.00	115.00	55	872.50	2643
Krull	1598.00	13216.50	1271.50	7271.25	1959.50	1001.50	3222	4229.60	8171
Kung Fu Master	258.50	40060.00	8925.00	13097.50	11755.00	9375.00	2090	14307.80	25900
Ms Pacman	307.30	2646.00	491.25	906.25	331.25	360.75	366	1465.50	1521
Pong	-20.70	21.00	2.17	-3.12	-20.85	-14.82	-20	-16.50	-4
Private Eye	24.90	105.10	47.50	-17.50	-85.00	-89.60	100	218.40	3238
Qbert	163.90	17148.75	644.38	1267.50	481.25	475.62	317	1042.40	2921
Road Runner	11.50	42895.00	10862.50	14950.00	972.50	6957.50	602	5661.00	19230
Seaquest	68.40	1874.00	839.00	861.50	309.50	720.50	305	384.50	962
Up N Down	533.40	282679.50	4679.50	2492.25	682.00	3245.25	1502	2955.20	46910
Ball in cup catch	421.37	736.60	449.20	147.62	787.48	0.00	829	970.00	972
Cartpole balance	355.11	999.94	999.79	500.76	916.33	139.48	516	980.00	993
Cartpole balance sparse	48.34	1000.00	1000.00	55.92	1000.00	17.80	881	999.00	964
Cartpole swingup	27.05	466.15	269.57	144.51	866.18	97.82	290	771.00	861
Cartpole swingup sparse	0.00	201.10	199.20	0.00	243.20	0.00	1	373.00	759
Cheetah run	3.69	366.96	277.22	109.16	272.25	247.72	95	502.00	836
Finger spin	0.52	398.85	198.48	136.85	425.40	1.32	118	880.00	589
Hopper hop	0.11	39.62	1.22	0.37	28.46	0.32	0	164.00	227
Hopper stand	4.76	526.23	45.29	35.91	70.19	3.49	4	777.00	903
Pendulum swingup	0.00	749.90	775.95	25.00	401.00	10.25	1	413.00	744
Reacher easy	68.70	420.20	64.50	50.62	150.88	106.40	487	689.00	951
Reacher hard	11.24	974.10	899.68	4.85	10.52	8.28	94	472.00	862

11.3 Algorithm scores correlation matrix

	My PPO	A2C	TRPO	R-PPO	PPO	CURL	Dreamer
My PPO	1.00	0.07	0.42	0.25	0.22	0.40	0.21
A2C	0.07	1.00	-0.04	0.37	-0.02	-0.24	-0.20
TRPO	0.42	-0.04	1.00	-0.00	0.47	0.75	0.35
R-PPO	0.25	0.37	-0.00	1.00	-0.04	-0.11	-0.29
PPO	0.22	-0.02	0.47	-0.04	1.00	0.53	0.47
CURL	0.40	-0.24	0.75	-0.11	0.53	1.00	0.68
Dreamer	0.21	-0.20	0.35	-0.29	0.47	0.68	1.00

Table 10: Correlation matrix for various RL algorithms.

11.4 Hyperparameters

Atari

PPO	A2C	Recurrent PPO	TRPO
n_envs: 8 n_steps: 128 n_epochs: 4 batch_size: 256 learning_rate: 2.5e-4 clip_range: 0.1 vf_coef: 0.5 ent_coef: 0.01 policy: CnnPolicy	n_envs: 16 n_steps: 5 gamma: 0.99 ent_coef: 0.01 vf_coef: 0.25 policy: CnnPolicy policy_args: optimizer_class=RMSprop, optimizer_kwargs: eps=1e-5	n_envs: 8 n_steps: 128 n_epochs: 4 batch_size: 256 learning_rate: 2.5e-4 clip_range: 0.1 vf_coef: 0.5 ent_coef: 0.01 policy: CnnLstmPolicy policy_args: enable_critic_lstm=False, lstm_hidden_size=128	n_envs: 8 n_steps: 2048 batch_size: 128 learning_rate: 0.001 gamma: 0.99 cg_max_steps: 15 cg_damping: 0.1 line_search_shrinking_factor: 0.8 n_critic_updates: 10 gae_lambda: 0.95 target_kl: 0.01 policy: CnnPolicy

Ball in cup catch, Reacher easy and Reacher hard

PPO	A2C	Recurrent PPO	TRPO
n_envs: 8 normalize_advantage: true batch_size: 32 n_steps: 512 gamma: 0.9 learning_rate: 0.000104019 ent_coef: 7.52585e-08 clip_range: 0.3 n_epochs: 5 gae_lambda: 1.0 max_grad_norm: 0.9 vf_coef: 0.950368 policy: MultiInputPolicy	n_envs: 8 normalize_advantage: false n_steps: 5 gamma: 0.99 gae_lambda: 1.0 ent_coef: 0.0 vf_coef: 0.5 rms_prop_eps: 1e-05 policy: MultiInputPolicy	n_envs: 1 normalize_advantage: true batch_size: 32 n_steps: 512 gamma: 0.9 learning_rate: 0.000104019 ent_coef: 7.52585e-08 clip_range: 0.3 n_epochs: 5 gae_lambda: 1.0 max_grad_norm: 0.9 vf_coef: 0.950368 policy: MultiInputLstmPolicy	n_envs: 2 normalize_advantage: true batch_size: 128 n_steps: 1024 gamma: 0.99 gae_lambda: 0.95 sub_sampling_factor: 1 cg_max_steps: 25 cg_damping: 0.1 n_critic_updates: 20 learning_rate: 1e-3 policy: MultiInputPolicy

Cartpole balance/swingup+sparse, Pendulum swingup, Finger spin

PPO	A2C	Recurrent PPO	TRPO
n_envs: 8 normalize_advantage: true batch_size: 64 n_steps: 32 gamma: 0.999 learning_rate: 0.000222425 ent_coef: 1.37976e-07 clip_range: 0.4 n_epochs: 5 gae_lambda: 0.9 max_grad_norm: 0.3 vf_coef: 0.19816 policy: MultiInputPolicy	n_envs: 8 normalize_advantage: false n_steps: 5 gamma: 0.99 gae_lambda: 1.0 ent_coef: 0.0 vf_coef: 0.5 rms_prop_eps: 1e-05 policy: MultiInputPolicy	n_envs: 1 normalize_advantage: true batch_size: 64 n_steps: 32 gamma: 0.999 learning_rate: 0.000222425 ent_coef: 1.37976e-07 clip_range: 0.4 n_epochs: 5 gae_lambda: 0.9 max_grad_norm: 0.3 vf_coef: 0.19816 policy: MultiInputLstmPolicy	n_envs: 2 normalize_advantage: true batch_size: 128 n_steps: 1024 gamma: 0.99 gae_lambda: 0.95 sub_sampling_factor: 1 cg_max_steps: 25 cg_damping: 0.1 n_critic_updates: 20 learning_rate: 1e-3 policy: MultiInputPolicy

Cheetah run

PPO	A2C	Recurrent PPO	TRPO
n_envs: 8 normalize_advantage: true batch_size: 64 n_steps: 512 gamma: 0.98 learning_rate: 2.0633e-05 ent_coef: 0.000401762 clip_range: 0.1 n_epochs: 20 gae_lambda: 0.92 max_grad_norm: 0.8 vf_coef: 0.58096 policy: MultiInputPolicy	n_envs: 8 normalize_advantage: false n_steps: 5 gamma: 0.99 gae_lambda: 1.0 ent_coef: 0.0 vf_coef: 0.5 rms_prop_eps: 1e-05 policy: MultiInputPolicy	n_envs: 1 normalize_advantage: true batch_size: 64 n_steps: 512 gamma: 0.98 learning_rate: 2.0633e-05 ent_coef: 0.000401762 clip_range: 0.1 n_epochs: 20 gae_lambda: 0.92 max_grad_norm: 0.8 vf_coef: 0.58096 policy: MultiInputLstmPolicy	n_envs: 2 normalize_advantage: true batch_size: 128 n_steps: 1024 gamma: 0.99 gae_lambda: 0.95 sub_sampling_factor: 1 cg_max_steps: 25 cg_damping: 0.1 n_critic_updates: 20 learning_rate: 1e-3 target_kl: 0.04 policy: MultiInputPolicy

Hopper hop and Hopper stand

PPO	A2C	Recurrent PPO	TRPO
n_envs: 8 normalize_advantage: true batch_size: 32 n_steps: 512 gamma: 0.999 learning_rate: 9.80828e-05 ent_coef: 0.00229519 clip_range: 0.2 n_epochs: 5 gae_lambda: 0.99 max_grad_norm: 0.7 vf_coef: 0.835671 policy: MultiInputPolicy	n_envs: 8 normalize_advantage: false n_steps: 5 gamma: 0.99 gae_lambda: 1.0 ent_coef: 0.0 vf_coef: 0.5 rms_prop_eps: 1e-05 policy: MultiInputPolicy	n_envs: 1 normalize_advantage: true batch_size: 32 n_steps: 512 gamma: 0.999 learning_rate: 9.80828e-05 ent_coef: 0.00229519 clip_range: 0.2 n_epochs: 5 gae_lambda: 0.99 max_grad_norm: 0.7 vf_coef: 0.835671 policy: MultiInputLstmPolicy	n_envs: 2 normalize_advantage: true batch_size: 128 n_steps: 1024 gamma: 0.99 gae_lambda: 0.95 sub_sampling_factor: 1 cg_max_steps: 25 cg_damping: 0.1 n_critic_updates: 20 learning_rate: 1e-3 policy: MultiInputPolicy

11.5 Regression R squares

Individual games predicting benchmark median scores, intercept enabled

Environment	R^2
Crazy Climber	0.000498
Freeway	0.014195
Assault	0.026691
Seaquest	0.040108
Demon Attack	0.045323
Breakout	0.055711
Pong	0.094365
Asterix	0.109813
Reacher easy	0.122651
Hero	0.146675
Krull	0.164081
Road Runner	0.183057
Jamesbond	0.260724
Gopher	0.266081
Amidar	0.285125
Battle Zone	0.322722
Alien	0.329899
Ball in cup catch	0.341246
Cartpole balance sparse	0.404618
Up N Down	0.407414
Boxing	0.428427
Kung Fu Master	0.466784
Private Eye	0.499432
Ms Pacman	0.513565
Chopper Command	0.517500
Reacher hard	0.551687
Qbert	0.556963
Cartpole swingup	0.620699
Cartpole balance	0.625066
Finger spin	0.664906
Cheetah run	0.675904
Frostbite	0.725132
Bank Heist	0.739152
Kangaroo	0.750271
Hopper hop	0.769898
Pendulum swingup	0.792666
Cartpole swingup sparse	0.836572
Hopper stand	0.926164

Table 11: R^2 values for various environments

Individual games predicting benchmark median scores, intercept disabled

Environment	**R^2**
Breakout	-2.419988
Demon Attack	-1.927779
Freeway	-1.647710
Crazy Climber	-1.298911
Boxing	-1.056587
Reacher easy	-0.740305
Up N Down	-0.671176
Krull	-0.658641
Pong	-0.475533
Jamesbond	-0.419280
Private Eye	-0.400845
Assault	-0.327148
Reacher hard	-0.206233
Kangaroo	-0.198346
Gopher	-0.182898
Ball in cup catch	-0.167255
Hero	-0.143747
Asterix	-0.098041
Battle Zone	-0.086580
Bank Heist	-0.052189
Seaquest	0.023900
Road Runner	0.024838
Alien	0.048907
Hopper hop	0.130556
Cartpole balance sparse	0.158818
Amidar	0.285044
Chopper Command	0.334152
Ms Pacman	0.386327
Kung Fu Master	0.414846
Cartpole swingup sparse	0.427358
Cheetah run	0.458224
Cartpole swingup	0.505481
Qbert	0.526478
Cartpole balance	0.610573
Hopper stand	0.617659
Finger spin	0.659333
Pendulum swingup	0.659774
Frostbite	0.722039

Table 12: R^2 values for various environments

Best 3 game subset predicting Individual game scores, intercept enabled

Environment	R^2
Jamesbond	0.338770
Freeway	0.397158
Up N Down	0.425836
Boxing	0.432482
Battle Zone	0.496734
Chopper Command	0.565000
Reacher hard	0.578207
Crazy Climber	0.615856
Cheetah run	0.672361
Hero	0.678408
Kung Fu Master	0.716362
Seaquest	0.723695
Cartpole balance	0.734337
Krull	0.749735
Private Eye	0.749739
Kangaroo	0.751569
Reacher easy	0.766022
Frostbite	0.771040
Gopher	0.791235
Finger spin	0.792321
Pong	0.801314
Hopper hop	0.810993
Qbert	0.814989
Bank Heist	0.840004
Cartpole balance sparse	0.886389
Road Runner	0.905645
Alien	0.913760
Breakout	0.913985
Assault	0.929180
Amidar	0.938481
Hopper stand	0.944157
Cartpole swingup	0.949614
Demon Attack	0.963003
Asterix	0.979652
Cartpole swingup sparse	0.992204

Table 13: R^2 values for environments

Best 3 game subset predicting Individual game scores, intercept disabled

Environment	R^2
Seaquest	-5.428782
Cheetah run	-2.484543
Kung Fu Master	-1.169213
Cartpole swingup	-0.556231
Amidar	-0.356898
Freeway	-0.264492
Assault	-0.009755
Crazy Climber	0.149939
Jamesbond	0.321285
Boxing	0.360029
Demon Attack	0.376932
Asterix	0.397199
Up N Down	0.400757
Battle Zone	0.431385
Breakout	0.468723
Road Runner	0.485052
Hero	0.490295
Reacher hard	0.517438
Chopper Command	0.562691
Pong	0.585305
Kangaroo	0.588383
Private Eye	0.647226
Bank Heist	0.676372
Cartpole balance	0.681383
Hopper hop	0.693349
Frostbite	0.748712
Krull	0.749585
Reacher easy	0.752688
Finger spin	0.759040
Qbert	0.769524
Gopher	0.782831
Hopper stand	0.816551
Cartpole balance sparse	0.865271
Alien	0.901890
Cartpole swingup sparse	0.940872

Table 14: R^2 values for environments