Pruning Performance on RL w/ Atari Games

List of Team Members:
Omar Yahia, 61853567, oyahia@uci.edu
Krishna Vempati, 11991922, vempatir@uci.edu

1. Introduction and Problem Statement

Machine learning models, particularly in reinforcement learning, have demonstrated remarkable capabilities in complex decision-making tasks, such as playing Atari games. However, these models often require significant computational and memory inefficiencies. This project explores these challenges by investigating a unique neural network pruning technique in the context of the Atari game Space Invaders. Specifically, we explore three types of pruning approaches, no pruning, random unstructured pruning, and L1 unstructured pruning, applied to two prominent reinforcement learning algorithms: Proximal Policy Optimization and Deep Q-Networks.

Our project specifically focuses on a novel approach of pruning, where we prune the neural network once, midway during the training process of the RL algorithms. This method was chosen due to its relatively unexplored nature in existing work, offering a new perspective to model optimization. By systematically analyzing how these pruning methods impact model performance, primarily measured through mean episodic return, we aim to develop insights on its effect towards reducing computational resource consumption while maintaining effective mean episodic return. This research is also significant for the broader field of artificial intelligence, as it directly addresses the growing need for more power-efficient and memory-optimized machine learning models, with potential implications for deploying advanced AI systems in resource constrained areas.

2. Related Work

The challenge of improving model efficiency in reinforcement learning has been a persistent research focus, with researchers continuously exploring methods to reduce computational complexity without significantly compromising model performance. While slow model running times have not completely halted RL algorithm development, the search for solutions to increase speed and potentially improve accuracy remains an active area of investigation. Previous research has primarily concentrated on various pruning techniques as a promising approach to address these computational challenges.

While existing studies have explored various pruning strategies, our research distinguishes itself through a unique approach. The research by [3], which examined Atari game environments, employed a dynamic pruning method with multiple pruning steps throughout training, in contrast to our single-step hybrid pruning approach. Similarly, the work by [4] took a fundamentally different pruning strategy by using Variational Autoencoders (VAE) to prune entire low-reward states, whereas our study focuses on network-level pruning techniques. The gradual magnitude pruning research provided a baseline for network optimization, but our work specifically investigates L1 and random unstructured pruning methods applied to PPO and DQN algorithms. By systematically comparing these three pruning techniques (no pruning, random unstructured, and L1 unstructured pruning) within a single hybrid pruning framework, our research aims to extend and nuance the existing understanding of pruning's impact on reinforcement learning models, particularly in the Space Invaders environment.

3. Data Sets

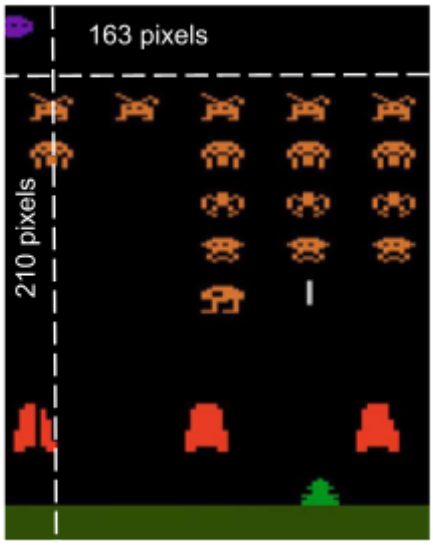
The dataset for this research is the SpaceInvadersNoFrameskip-v4 environment, sourced from the OpenAI Gymnasium (https://ale.farama.org/environments/space_invaders/). We made the decision to disable frame skipping to avoid visual artifacts, particularly with alien bullet rendering in the game. We also chose not to use the RAM-based observation space to minimize observation complexity, focusing instead on the visual representation of the game state.

Environment Specifications:

- **Observation Space**: RGB image with dimensions 210x163 pixels
- **Action Space**: Six discrete actions:

Do Nothing	**Move Right**	**Move Left**
Fire	**Move Right and Fire**	**Move left and Fire**

Image Source - [9]



*Preprocessing Pipeline***:** Our data preprocessing involves three key transformations to prepare the input for our reinforcement learning models:

1. Convert RGB image to grayscale (black and white)
2. Downsample image to 84x84 pixels
3. Create observation stacks by combining 4 consecutive frames

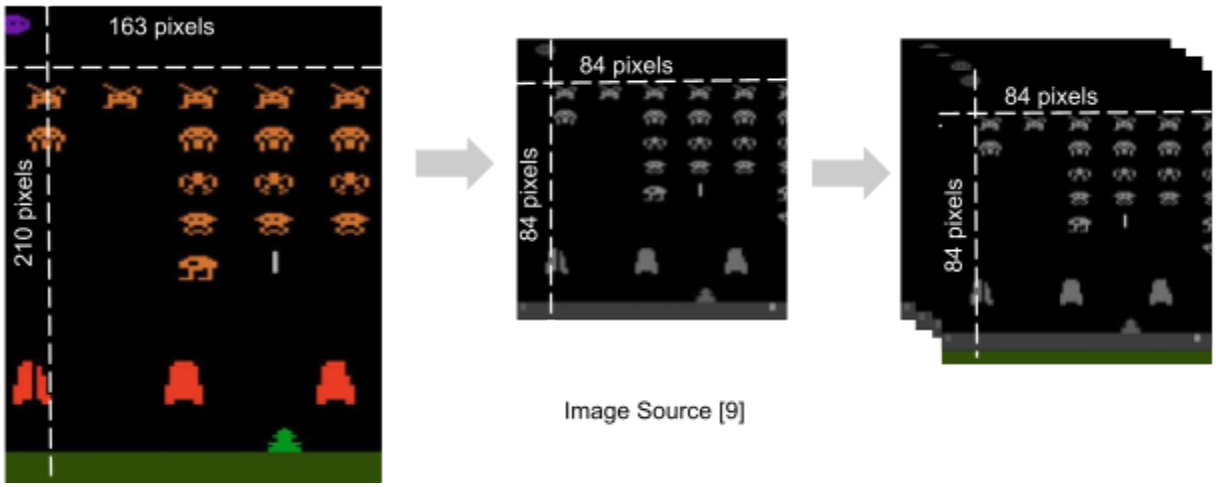


Image Source [9]

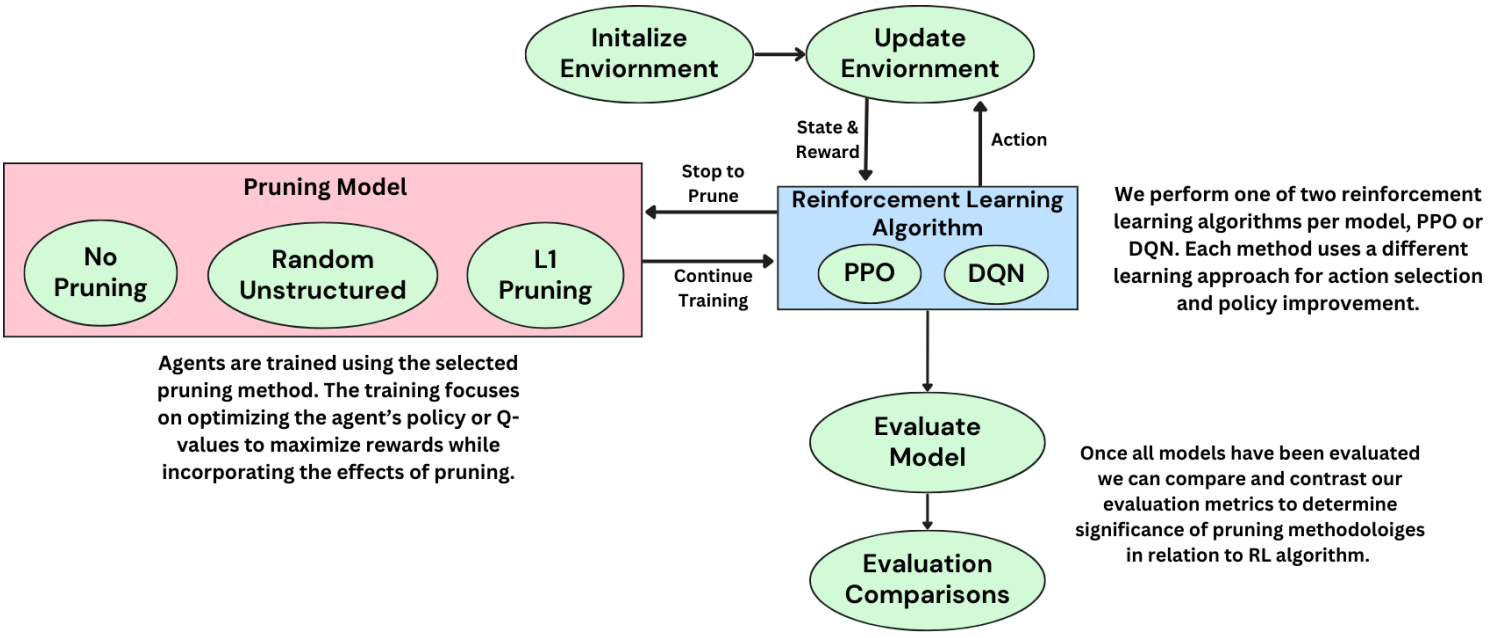
This preprocessing approach standardized the input, reduced computational complexity, and provided temporal context by stacking frames, which is crucial for capturing motion and game dynamics in our rl algorithms.

4. Description of Technical Approach

Our project focused on creating six distinct models: two for each pruning type (L1, none, and random), with one PPO and one DQN model for each pruning category. Deep Q-Networks and Proximal Policy Optimization are two distinct reinforcement learning algorithms. DQN approximates Q-values using neural networks to predict action rewards, utilizing experience replay and target networks for stable learning. PPO is an actor-critic method that optimizes a stochastic policy with constrained updates, balancing exploration and exploitation effectively. Using distinct algorithms allowed us to test the relative effectiveness of pruning on them.

We employed two pruning techniques: random and L1 unstructured pruning. Random pruning removes weights randomly, while L1 pruning removes weights based on their L1 norm magnitude. These methods can potentially reduce model complexity, decrease computational requirements, and potentially improve generalization by removing less important weights. For each method of pruning we only prune the weights and biases but not the entire neuron in a model.

Our pruning approach is unique: we implemented a hybrid pruning method by pruning the network only once halfway through training. This differs from both static pruning (pruning after training) and dynamic pruning (multiple pruning steps throughout training). The pruning also was performed with various percentages: for DQN, we used L1 and random pruning at 0.05, 0.1, 0.3, 0.5, and 0.8 weight removal rates; for PPO, we used 0.1, 0.3, and 0.5 weight removal rates. For example if a model ran with 0.3 weight removal rate, then during the middle of training 30% of weights will be removed and changed to 0s.



We leveraged the CleanRL library as our primary implementation framework, building upon its base PPO and DQN implementations. We extended these implementations by adding custom pruning steps and evaluation metrics, including memory usage tracking for tensor models. PyTorch served as the core algorithm implementation platform, providing robust deep learning capabilities, while NumPy supported data storage and mathematical operations. We used

OpenAI Gymnasium for the Space Invaders environment and developed our research in Visual Studio Code.

A key technical challenge was reimplementing evaluation code for PPO, as the original CleanRL implementation only included evaluation for DQN. We also noted that PyTorch does not physically remove weights during pruning, instead using weight masking to simulate pruning, as a fully sparse tensor would reduce hardware performance. Another challenge we faced was showing the positive effects of pruning by lowering memory usage. This was because inherently the matrices in pytorch were implemented to mask pruning as changing weights to 0 instead of actually omitting data from a model. To combat this challenge we showed the storage space taken manually after pruning.

Our development process broken into steps:

1. Experimenting with the CleanRL repository
2. Modifying existing implementations for DQN and PPO
3. Adding Pruning implementations
4. Creating shell scripts to test various model combinations
5. Using Weights & Biases for run tracking and performance evaluation

5. Software

(a) Code we have written:

For DQN and PPO, we wrote code to integrate the pytorch pruning library and evaluate the resulting sparsity (percentage of total weights that are zero). For integrating the pruning library, this included modifying the Cleanrl implementation to add new parameters to enable pruning and also specify the pruning type and amount. For evaluating the sparsity, this was done in two parts: first, we created code to count the number of non-zero weights. Second, specifically for DQN we experimented with saving the weights as a pytorch sparse tensor for every layer with weights in the network in order to see the space saved on disk. We also wrote code fixing Cleanrl's PPO evaluation which was outdated and incompatible with their current Atari PPO code. Finally we wrote shell scripts to repeatedly test different pruning parameters and upload the resulting data to the weights and biases platform.

(b) Code from other people that we used:

We used Python 3.9 as our programming language, alongside PyTorch compiled with CUDA hardware acceleration to leverage GPU enhancements for efficient deep-learning computations. The base code for implementing PPO and DQN algorithms was built upon the CleanRL library, which provided a baseline for reinforcement learning implementations, including functionalities for evaluation metrics and mechanisms for saving and uploading trained models. Environment interaction was managed using OpenAI's Gymnasium API, along with the Farama Foundation's Arcade Learning Environment for running Atari-based environments. Video recording of agent performance was achieved through the integration of FFmpeg, while various environment wrappers—such as those for reward clipping—were implemented using Stable Baselines3.

For experiment tracking, logging, and hosting models and evaluation metrics, we utilized the Weights and Biases (W&B) platform, enabling online, real-time visualization of results. CleanRL's reliance on Tyro further enhanced the project by providing a clean and flexible interface for command-line argument parsing. Additionally, PyTorch's torch.nn.utils.prune module and its support for sparse tensors were employed to prune neural network weights.

6. Experiments and Evaluation [at least 1 page]

(a) Experimental Setup

To evaluate the performance of our pruning techniques, we used the mean episodic return from evaluation episodes as the primary metric. The episodic return represents the total reward a model accumulates during a specific episode. Given the variability in rewards during training due to the trade-off between exploration and exploitation, we recorded the mean episodic return only during evaluation. For evaluation, we averaged the episodic return over ten episodes to ensure consistency and reliability in our results.

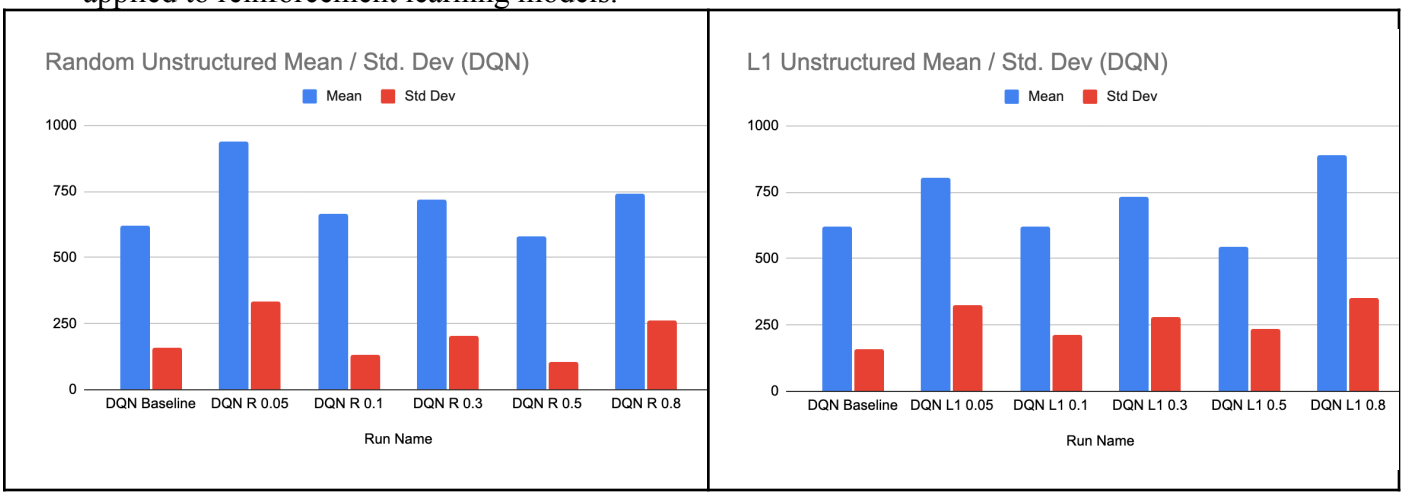
In addition to the mean episodic return, we analyzed the performance of sparse representations through pruning by examining the following metrics:

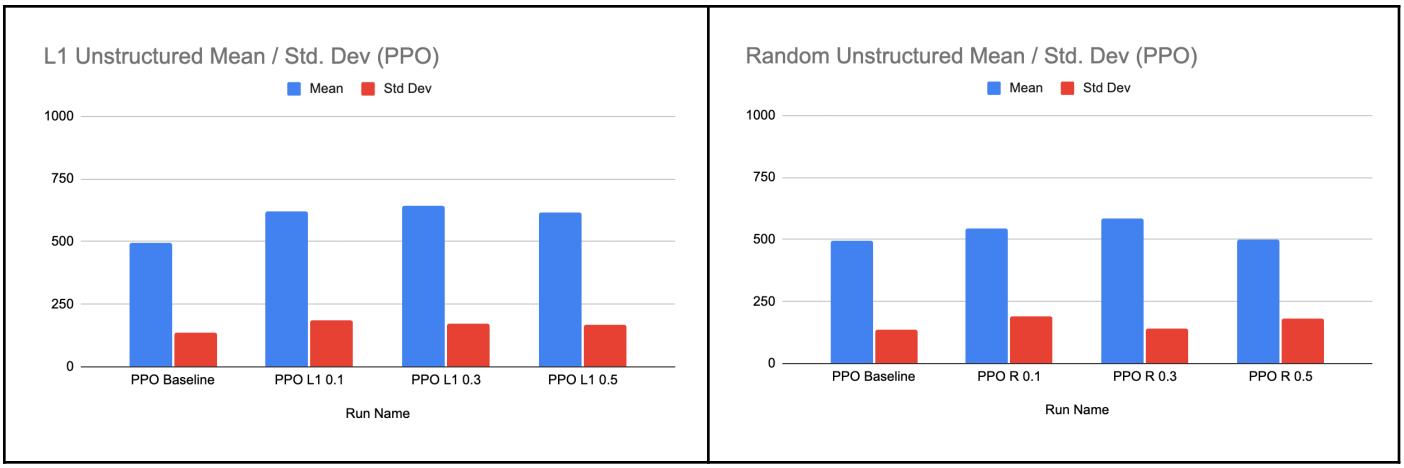
1) Non-zero parameters in the weight matrix: This metric helped quantify the sparsity introduced by pruning.
2) Storage size of sparse tensors on disk: This allowed us to estimate the memory efficiency achieved through pruning.
3) Steps per second: By plotting the steps per second, we could evaluate the speed of convergence for each model.

Note: All models were run for 2 million timesteps to maintain consistency across experiments and provide comparable results.

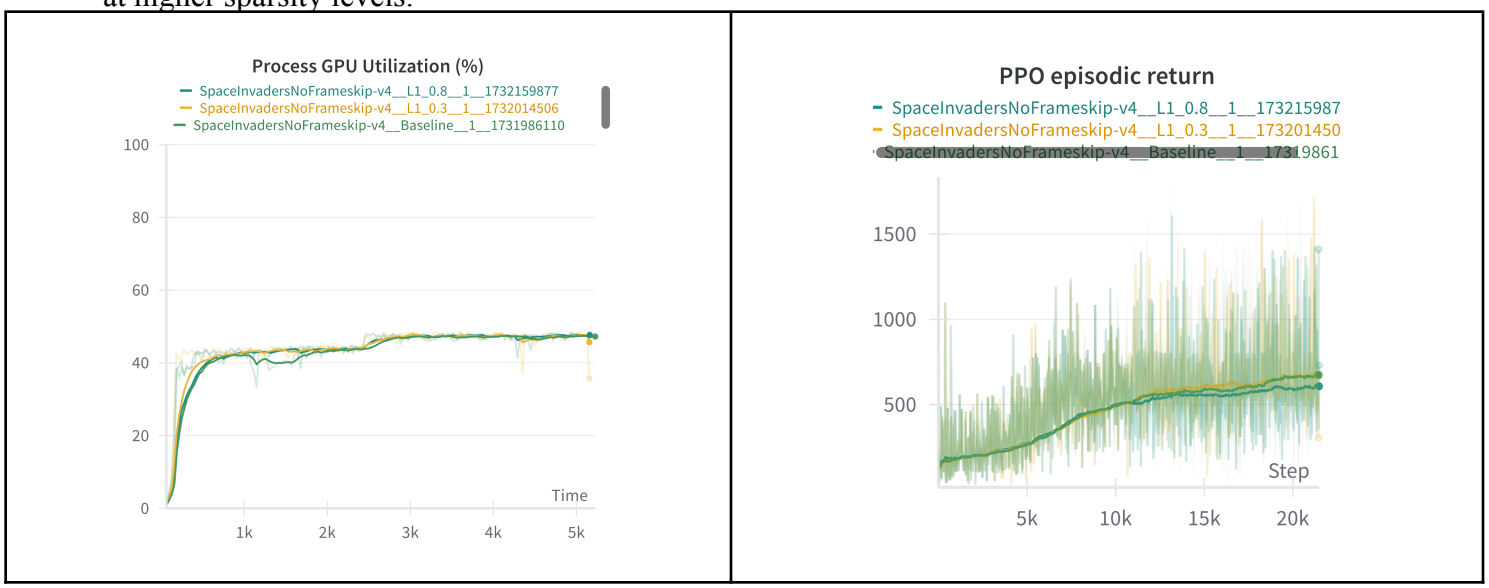
(b) Results and Observations

Below, we present our final results and insights based on the analysis of pruning techniques applied to reinforcement learning models.

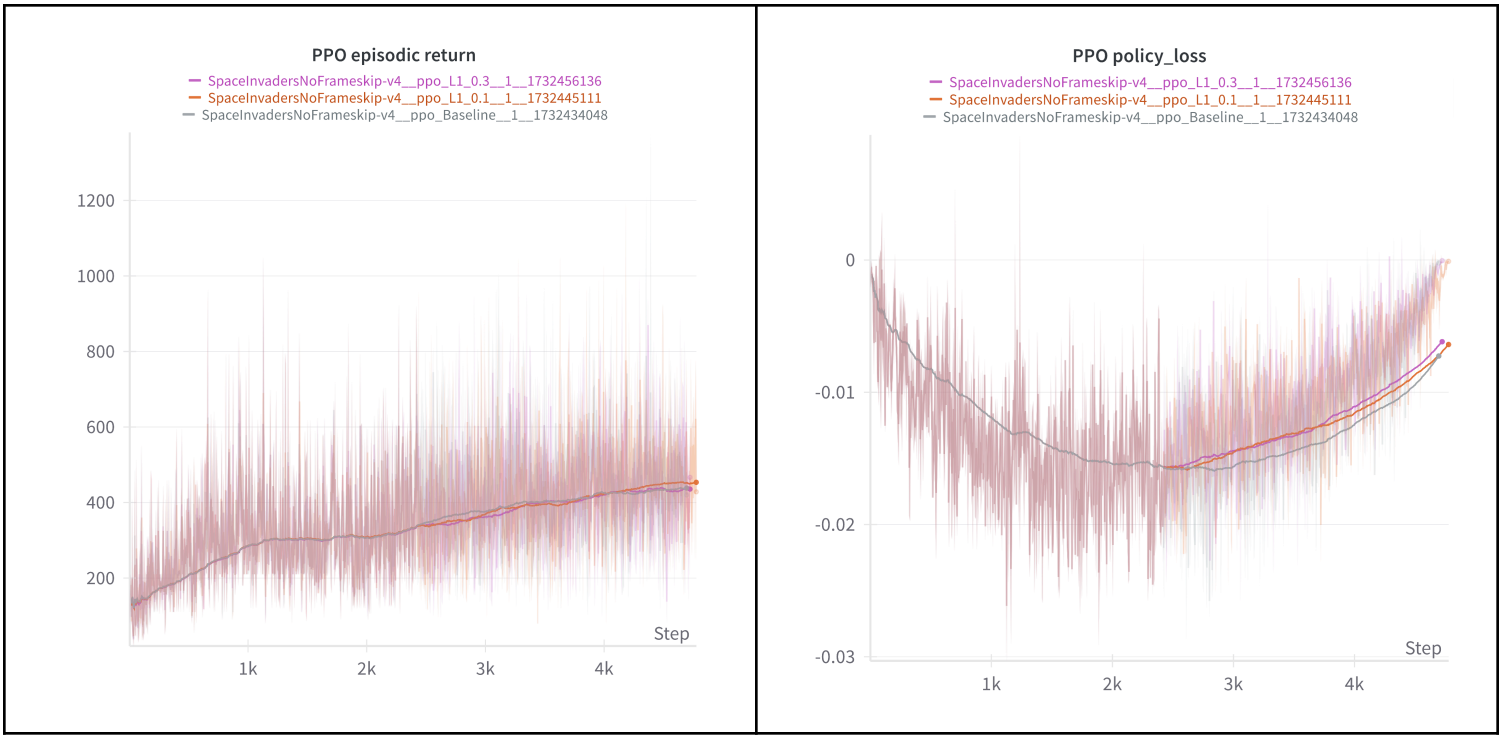




The above images show that pruning had minimal impact on evaluation performance. Models with a high mean episodic return also exhibited high standard deviation, regardless of whether random or L1 unstructured pruning methods were used. This trend was consistent across different pruning percentages, suggesting that pruning maintained evaluation performance even at higher sparsity levels.



Above the graphs show that the GPU utilization was largely unaffected by pruning. However, pruning above 30% resulted in lower episodic returns during training. This indicates a potential threshold beyond which pruning begins to negatively influence the model's ability to learn effectively.



The episodic return showed little to no difference across pruned and unpruned models. Similarly, the policy loss trends were stable, with exponential moving average lines returning to baseline loss values over iterations. This suggests that pruning did not significantly impact the training dynamics of the models.

Overall, our experiments demonstrated that pruning can be a viable method for improving model efficiency without drastically affecting performance, as long as the pruning percentage remains within reasonable bounds.

7. Discussion and Conclusion

We learned from the project that the method of pruning, type of pruning, and the model architecture can all affect how well rl algorithms perform. Different strategies had varying effects, but we found that all the pruned models performed almost as well as the unpruned ones. This means that as long as pruning is done carefully and not too aggressively, it doesn't necessarily harm the model's performance. It also suggests that many of the weights in neural networks may not be essential, and removing some can improve efficiency without a big loss in accuracy. This makes pruning a useful technique for saving resources while maintaining good results.

One limitation we faced was that we couldn't measure how pruning affected memory usage because PyTorch doesn't fully support sparse matrices. Sparse matrices, which take advantage of the zeros introduced by pruning, could have made the models even more efficient. In the future, it would be worth exploring libraries or tools that support sparse computations to better understand the memory benefits of pruning. It would also be interesting to compare pruning in models with different starting sizes to see how removing weights impacts smaller models. Additionally, looking at how pruning interacts with dropout layers could help us understand how these methods work together to prevent overfitting. These ideas provide useful directions for building on what we've learned.

8. Appendix and Bibliography

1) [1] Berner, C., Brockman, G., Chan, B., Cheung, V., Psyho, quot;, Dębiak, quot;, Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., Pondé, H., Pinto, O., Raiman, J., & Salimans, T. (2019). Dota 2 with Large Scale Deep Reinforcement Learning. https://cdn.openai.com/dota-2.pdf
2) [2] Kindig, B. (2024, June 20). AI Power Consumption: Rapidly Becoming Mission-Critical. Forbes. https://www.forbes.com/sites/bethkindig/2024/06/20/ai-power-consumption-rapidly-becoming-mission-critical/
3) [3] Kaloev, M., & Krastev, G. (2023). Comprehensive Review of Benefits from the Use of Neuron Connection Pruning Techniques During the Training Process of Artificial Neural Networks in Reinforcement Learning: Experimental Simulations in Atari Games. 2023 7th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT), 1-6. https://doi.org/10.1109/ISMSIT58785.2023.10304968.
4) [4] Ayton, B., & Asai, M. (2021). Width-Based Planning and Active Learning for Atari. ArXiv.
5) [5] Obando-Ceron, J., Courville, A., & Castro, P. S. (2024). In value-based deep reinforcement learning, a pruned network is a good network. https://arxiv.org/abs/2402.12479
6) [6] Omar and Lorraine. (2018, September 7). How does Space Invaders on the Atari 2600 display all the aliens? Retrocomputing Stack Exchange. https://retrocomputing.stackexchange.com/questions/7502/how-does-space-invaders-on-the-atari-2600-display-all-the-aliens
7) [7] Tensorflow. (2019). TensorFlow Model Optimization Toolkit — Pruning API. Tensorflow.org. https://blog.tensorflow.org/2019/05/tf-model-optimization-toolkit-pruning-API.html
8) *Space Invaders Atari*. (n.d.). https://i.sstatic.net/rmN5u.jpg