Model Compression and Fine-Tuning for Reinforcement Learning

Nidhir Guggilla, Andrew Kim, Kaushik Kunal Singh, Gaurav Bhatnagar

December 16, 2023

Model compression in large neural networks is used to reduce the computing power, memory, and time required to run those networks while still maintaining comparable performance. However, not much research has been done into the performance of model compression methods when used for reinforcement learning - probably because most traditional reinforcement learning methods don't rely on extremely large neural networks. Despite this, we hypothesized that compression will still improve run-time, working set size, and CPU usage during inference while causing acceptable reductions in performance. We tested two types of baseline models, training them with policy gradients and double Q-learning respectively. We trained and evaluated these agents on OpenAI Gym environments, with policy gradients used for the Half-Cheetah environment while double Q-learning was used for LunarLander. We compressed the models using low-rank approximation, weight magnitude pruning, and quantization. We also tested fine-tuning methods that take advantage of the two network structure of double Q-learning in order to further mitigate the performance losses caused by compression.

Our fine-tuning methods for pruning and low-rank approximation for double Q-learning involved compressing the primary Q network at regular intervals during training but maintaining the target Q network at full size. Evaluation was performed using the compressed primary Q network. We found that compression improved our efficiency metrics (CPU usage, RAM usage, inference time), with pruning having a larger effect than low rank approximation. We also showed that fine-tuning is able to significantly improve the performance of compressed models compared to their naive counterparts, notably at a higher sample efficiency than simply training a smaller model from scratch.

Keywords: model compression, low-rank approximation, pruning, quantization, fine-tuning, measurement.

1 INTRODUCTION

1.1 *Problem Statement*

Our research addresses the challenge of handling increasingly large RL models, which can be computationally demanding and memory-intensive. These networks may not be suitable for deployment on resource-constrained devices, such as embedded systems, IoT devices, or other edge devices. In order to tackle this problem, we aim to identify model compression techniques that reduce the computational and memory requirements of these models while maintaining acceptable performance.

Furthermore, some devices lack the necessary hardware support to perform the required operations for reinforcement learning, especially bare-metal computers which lack specifically Floating Point Units (FPUs). This limitation extends to basic operations like floating-point division. To make neural networks compatible with such devices, we explore quantization, which converts model weights and biases from floats to smaller integers that satisfy bit width constraints, while maintaining performance.

1.2 *Literature Review*

In our exploration of related work, we delved into two primary aspects of this project: compression techniques applicable to machine learning models in a broader context, and model compression specifically tailored to reinforcement learning.

In the realm of compression techniques for machine learning models, we encountered a rich body of prior research that encompasses a wide spectrum of methodologies. These encompass techniques such as pruning, low-rank approximation, quantization, distillation, and binarization, among others. A comprehensive survey of these methods can be found in Neill's work [**Neill2020**], which offers an extensive overview of the field.

Turning our attention to model compression within the domain of reinforcement learning, we observed that the existing body of literature is comparatively narrower. This is likely attributed to the characteristic use of compact neural networks in RL algorithms, which inherently require less compression. Nonetheless, our search did yield a handful of papers that address this specific subject matter, with notable examples being Krishnan et al.'s investigation [**Krishnan2022**] and García-Ramírez et al.'s exploration [**GarciaRamirez2022**]. However, it is worth noting that these aforementioned papers primarily emphasize the performance evaluation of compressed RL models in terms of their achieved returns. Our focus, in contrast, extends towards a more profound examination of the computational intensity aspect of model compression in the context of RL.

1.3 *Paper Organization*

The structure of our paper is organized as follows. We begin by discussing the methods employed in our research, including data measurement and model compression techniques. Next, we present the results of our experiments, followed by a detailed discussion of the findings. Finally, we conclude our paper with limitations of our work, suggest directions for future research and a summary of our contributions.

2 METHODS

2.1 *Data Measurement*

In our experimentation, we employed a Ryzen 7 2700x CPU and an Nvidia 1660 Ti GPU as our hardware foundation. To facilitate the training and evaluation of our reinforcement learning models, we developed a wrapper software that concurrently gauged system resource consumption.

With a focus on constrained compute and specialized devices, we took the approach of isolating a single core on our CPU, ensuring that all training and evaluation scripts ran exclusively on this core. This deliberate isolation streamlined our CPU measurement statistics. Additionally, we integrated process statistic software into our measurement script, enabling frequent polling of CPU and RAM usage statistics. We designed certain computations to be GPU-bound and utilized the Nvidia management library to measure power usage and utilization rates for the GPU.

Notably, our GPU power measurements initially raised suspicions, as we struggled to see any trends. However, we attribute this anomaly to non-differentiated GPU usage in our original compression methods, as we were able to generally validate the precision and accuracy of our measurement methods. To do this, we applied them to tasks of known computational complexity levels, such as matrix multiplication and statistical analyses on the resulting matrices. Through this process, we found tractable correlations between task complexity and resource usage for every tracked statistic.

We ran these measurements on both training/fine-tuning and evaluation. We trained and fine-tuned the models for 300000 steps. We evaluated the models across 100 episodes, which consist of the model controlling the agent until either failure or success. We measured average CPU, GPU, and RAM usage, GPU power draw, total time, episode length, and agent performance across the 100 episodes. We calculated the time it took for each model to perform inference (eval time) by dividing the total time measurement by the total episode length (the number of actions taken in each episode summed across all 100 episodes).

2.2 *Baseline Models*

The majority of our experiments were performed on an agent trained using double Q-learning for the OpenAI LunarLander environment. The LunarLander task is a discrete environment in which a shuttle's thrust engines have to be appropriately controlled to land the ship between two flags on a surface. Q-learning is a form of reinforcement learning where a neural network is trained to predict the reward associated with each action an agent can take. Double Q-learning expands on this by introducing a second network (the target network) in order to produce more accurate predictions of reward and reduce variance. The networks we tested our methods on had two hidden layers, each with size 64. We also trained agents with hidden layer sizes of 32 and 16 in order to be able to compare the effects of post-training compression with reducing the size of the model before training (ie. pre-training compression).

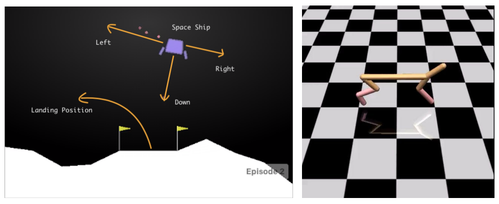

Figure 1: LunarLander and Half-Cheetah

We also ran a smaller subset of experiments on the Half-Cheetah task, which is a continuous environment in which the policy has to control a 2-dimensional figure with one front leg and one back leg. We experimented with policy gradient, which is a reinforcement learning scheme in which gradient descent is directly applied to the reward function of the decision-making policy. Specifically, we evaluated the performance of quantizing the mean_net and logits_net of the actor.

2.3 *Model Compression*

The three forms of model compression we implemented for our experiments were pruning (sparsification), low rank approximation, and quantization.

Pruning (Sparsification): Pruning a neural network involves removing some of the connections between layers by changing the corresponding weights to zero or entirely removing nodes. This aims to reduce model size and inference time by reducing the number of weights that a given model needs to store and use for inference. For example, a prune level of 0.4 means that 40% of the weights in the model were zero, with the remaining weights being the ones of highest magnitude in the original model. We implemented pruning of weights by changing the ones with the smallest magnitudes to zero using the PyTorch prune utilities.

Low Rank Approximation: Matrices typically take up a quadratic amount of space in their dimension. An N by N matrix is populated by N^2 entries. Taking the singular value decomposition of a matrix is a form of Fourier transform in which the underlying dyad matrices can be revealed. These dyad matrices can be ranked by how much of an effect they have on the composition of the original matrix (in a number called the singular value), and can also be represented as the product of two N-length vectors. LRA is an approximation technique that disregards the composing matrices with the lowest singular values and re-composes with the remaining ones. The long-term space required to store would now grow as $2N * P$, where P is the number of composing matrices you keep. We implemented this using the PyTorch SVD module to select the dyads corresponding to the largest singular values and replace the weight matrices in place. We tested two forms of low-rank approximation: one where we only compressed the hidden layers of the network and one where we compressed every layer (including the input and output layers). We chose to analyze the results of only performing low-ran approximation on the hidden layers of the network because the initial rank of the input/output layers was already so small that any amount of rank reduction made it nearly impossible for them to perform at a reasonable level while also leading to enough decreases in memory and CPU usage that they drowned out any other signals.

Quantization: Quantization fundamentally works by mapping continuous floats to discrete integers (we focused on 32-bit floats to 8-bit ints), and we explored two main methods of this: Quantization Aware Training (QAT) and Post Training Quantization (PTQ). QAT involves training the model while simulating the effects of quantization through quantization and dequantization nodes. PTQ, on the other hand, applies quantization after the model is alreadry trainedWe implemented these approaches using Pytorch's quantization library (both in 'eager mode' and 'fx mode'). Our results, however, focus on the impact of QAT on our benchmarks, as we were interested in retaining the performance of the uncompressed model.

We also attempted binarization as an extreme form of quantization, but faced challenges in implementing it effectively for our application. Its experiments, therefore, have not been included.

2.4 *Fine-Tuning*

We implemented post-training fine-tuning with the goal of compensating for the decrease in return caused by compression while maintaining the benefits in efficiency. Fine-tuning generally involves starting with a model that has been trained on one task or dataset and further training it for a different setting, potentially freezing some weights and using a different learning rate to allow the model to remember its earlier training while still being able to adapt to the new scenario.

Our method focused on taking advantage of the two network structure of double Q-learning to help the agent cope with the loss of expressivity caused by compression. We ran two sets of fine-tuning experiments. In one, we trained the primary Q-network normally while compressing the target Q-network at regular intervals. In the other, we ensured that we transferred an uncompressed version of the primary Q-network to the target network before compressing the primary Q-network at regular intervals. Our intention with the first experiment was to see if a fully expressive primary network with a compressed target network would be able to eventually learn a representation that would compress well (motivated by the targets provided by the compressed target network), improving performance after fine-tuning. The goal of the second experiment was to see if a compressed primary network would still be able to learn to predict the more accurate targets being provided to it by the uncompressed target network.

3 RESULTS

3.1 *Graphs*

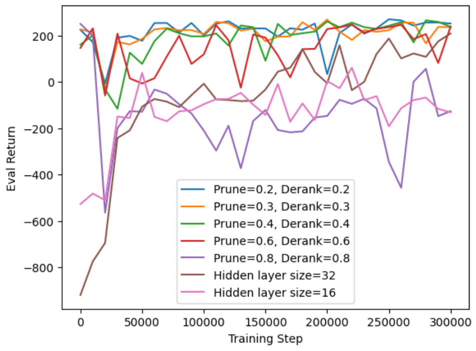

Figure 2: Training curves for fine-tuning models after compression, as well as smaller models from scratch

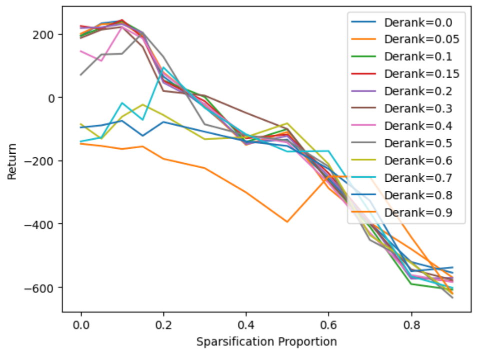

Figure 3: Return achieved by naive compression as a function of pruning amount (sparsification)

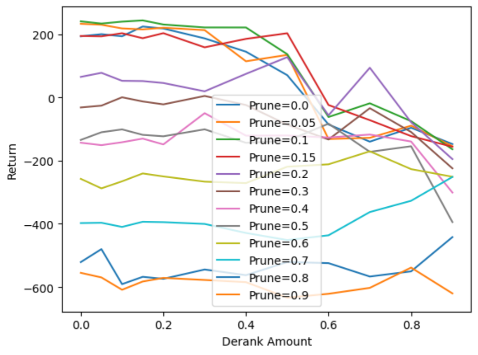

Figure 4: Return achieved by naive compression as a function of low-rank approximation amount (derank) when only reducing rank of hidden layers

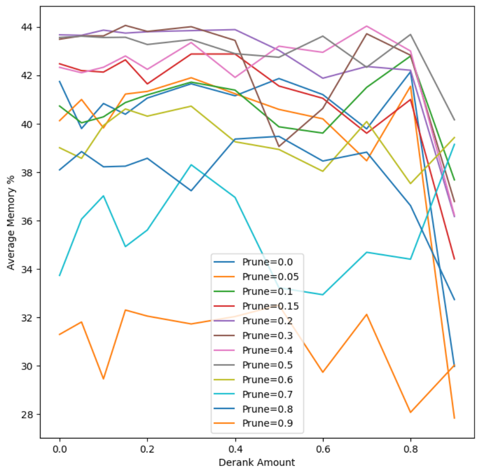

Figure 5: Average memory used as a function of low-rank approximation amount (derank)

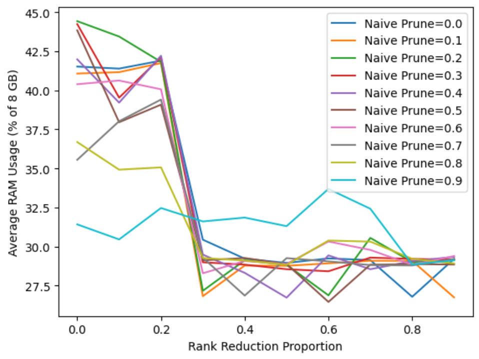

Figure 6: Average RAM used by naive compression as a function of low-rank approximation amount (derank) when also reducing rank of input/output layers

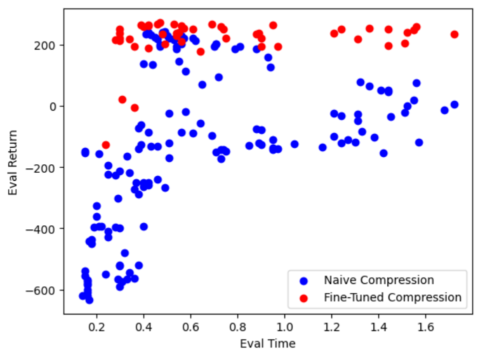

Figure 7: Return achieved as a function of time to perform 100 transitions (Eval Time) for naively compressed and fine-tuned models

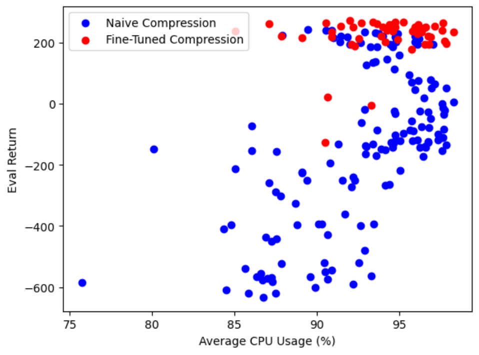

Figure 8: Return achieved as a function of average CPU usage during 100 transitions for naively compressed and fine-tuned models

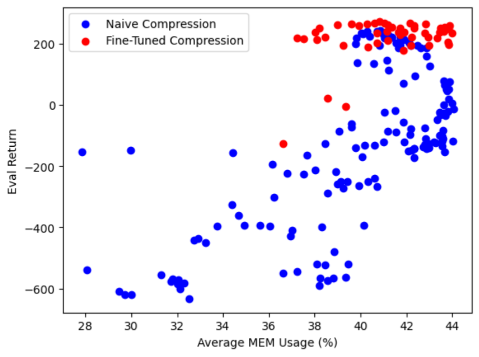

Figure 9: Return achieved as a function of average RAM usage (Average MEM Usage (%)) during 100 transitions for naively compressed and fine-tuned models

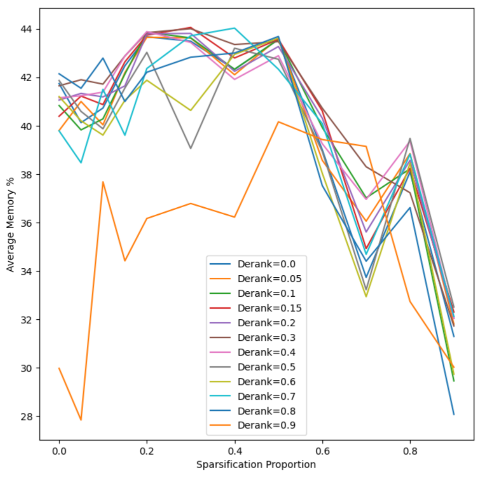

Figure 10: Average RAM used by naive compression as a function of amount pruned from the network (sparsification proportion)

3.2 *Analysis*

Figure 2 shows training curves for agents pre-trained with double Q-learning with hidden layer size 64 in the LunarLander environment that are then compressed and fine-tuned. In this graph, we examine the training curves of some specific compression levels to highlight how long they take to recover from the effects of compression as we fine-tune them. All of the compressed models show a distinct drop in performance at 20000 steps because until this point, the uncompressed model was being used to collect data for fine-tuning. At 20,000, performance drops to the level it would be at for naive compression and further steps are used to recover. We also include the training curves for smaller hidden layer sizes (32, 16) to compare the amount of training required to fine tuning a compressed model instead of training a smaller model from scratch. We can see that hidden layer size 32 (the brown curve) is able to eventually reach the performance of our compressed and fine-tuned models, but it takes almost 300,000 training steps to do so. In comparison, 'low' compression amounts (pruning and deranking under 0.5) are able to recreate their initially rich performance within 75,000 training steps. Finally, we see that a hidden layer size of 16 is unable to reach performance near 200, plateauing early around a return of -150, similar in performance to a simultaneous deranking and pruning of 0.8. In cases where a pre-trained model already exists but is too large for a specific use case, it takes less training to compress and fine-tune the existing model than attempt to train a smaller model from scratch. In reinforcement learning use cases where a system may have to adapt on the fly and learning time is vital, we suggest that compression techniques as used on prior trained models may be more efficient than using smaller models to begin with.

Figures 3 and 4 were evaluations of LRA and pruning techniques in various degrees on agents trained with DQL for LunarLander. As we can see from figure 3, the sparsification had a highly tractable and negative effect on performance of the actor. This was noticeable in all levels of deranking, from 0 to 90%, although derank amounts under and including 50% were generally in a higher class of performance which approximately reproduced the performance of the unaltered base model in its reactivity to sparsification. Figure 4 supports this conclusion, with the higher-performing models (those with lower pruning amounts) staying relatively stable around their starting return values until reaching 50% deranking, at which point they dropped dramatically. We can see from these two graphs that weight pruning has more tractable effects on return than deranking. We also paradoxically noticed that, for low derank models, going from 0% to 5% or even 10% weight pruning was able to generally improve performance of the model. We attribute this to possibly being due to some form of overfitting mitigation. On the other hand, when p

Figures 17 through 9 display performance over CPU resource statistics and evaluation time, all in the case of solely inference (decision making). The red points represent the final performance of compressed models after repeated fine tuning and re-compression. The general trend of red nearing the top of the graphs with corresponding blue points directly underneath indicates that, for similar levels of CPU usage and MEM (RAM) usage, fine tuning is able to achieve better return, e.g. make better use of resources. Within the naively compressed models, we can generally see positive relationships between resource usage and performance. As our power utilization statistics were stable across our experimentation, we can also view evaluation time as a proxy for energy expended during inference.

Figure 9 validates the actual utility of our compression techniques, as we see a negative correlation between sparsification and RAM usage in our CPU. Lower degrees of weight pruning failed to show this relationship, but it is evident at higher proportions.

We tested two forms of low-rank approximation, one where only the hidden layers of the network were compressed and one where every layer in the network was compressed. Figures 5 and 9 show the results when using LRA only on the hidden layers while figure 6 shows the impact of deranking all layers of the network. We can see that compressing the input and output layers has a drastic effect on the model, as evidenced by the sudden drop off at 0.3 sparsification, whereas 5 remains relatively stable. Changing the rank of the initially low-rank input and outputs significantly damages expressivity of the model.

3.3 *Quantization analysis*

In most of our experiments, we did not notice a significant difference in average cpu utilization, RAM utilization or other performance benchmarks after compressing the model with in-training quantization. We hypothesize that this could be due to several reasons:

1) The addition of quantize nodes in the computation graph, adding additional computational steps for each weight and bias, which can be seen by figure 11.

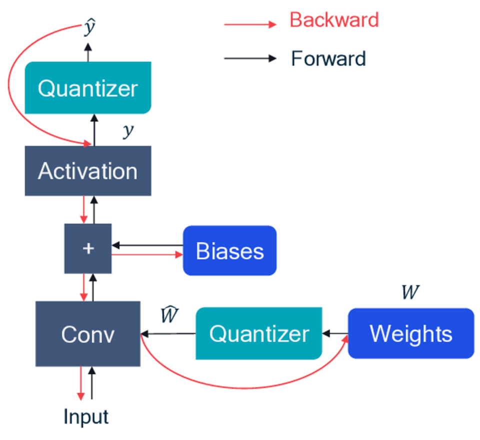

Figure 11: Computation graph for Quantization

2) The PyTorch quantization library we employed is encapsulated within a black-box framework and may be primarily optimized for optimizing return performance, possibly at the expense of other performance benchmarks we were assessing.

However, we did notice an expected decrease in stored size of the model weights, from 0.3MB to 0.1MB. Additionally, we noticed a trend in training time when compared to no compression and pruning, as seen in figure 12.

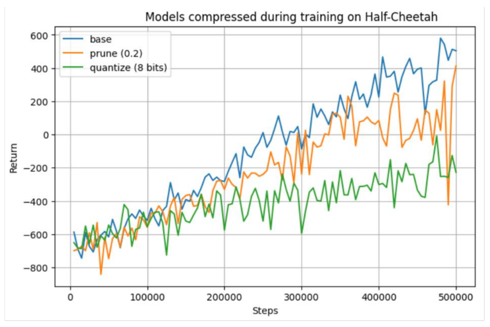

Figure 12: Training times

4 CONCLUSION

4.1 *Limitations*

Our project's scope was influenced by the utilization of an existing quantization library, which constrained us to the implementation and analysis of a specific quantization level, specifically, the 32-bit to 8-bit quantization. Ideally, we would have preferred to explore various quantization levels and assess their impact on model performance, as well as resource utilization metrics.

It's important to note that despite our rigorous sanity-checking efforts, we encountered challenges in discerning notable trends in GPU power consumption. This limitation stemmed from relatively low GPU utilization levels, consistently below 10 percent, throughout the majority of our experimentation. This discrepancy may be attributed to the fact that we averaged these values across the entirety of the training and evaluation loops, rather than focusing solely on the backpropagation and inference segments, which could have yielded more insightful GPU power statistics.

Additionally, our non-quantization compression methods employed abstracted proportions represented as arbitrary decimals. However, it's essential to recognize that beneath the abstraction, the removal of weights and singular values necessitated the handling of integer quantities, introducing a level of complexity that warrants consideration.

4.2 *Future Work*

In the realm of future developments, one promising avenue is the implementation of a custom interface designed for the storage of a model's weights and biases, along with the execution of pertinent operations on them. This tailored interface has the potential to unlock the full spectrum of performance benefits that model compression techniques can offer. These enhancements could include the introduction of vectorized dynamic bit widths and memory defragmentation strategies, among others, to further optimize resource utilization.

Furthermore, our experiments primarily adhered to a fixed set of hyperparameters during the fine-tuning process. In subsequent research endeavors, the exploration of varying hyperparameter configurations, such as learning rates, presents an opportunity to fine-tune compression methods to achieve even better results. Of particular interest is the investigation into the effects of altering the frequencies at which model compression and copying to the target network occur, potentially unveiling novel optimization strategies.

To refine the precision of our results, a more sensitive and granular approach to measuring power consumption across specific segments of the training and inference processes could be adopted. Additionally, parameterization in more suitable units may lead to more precise and informative outcomes in terms of resource utilization and performance evaluation.

4.3 *Final Thoughts*

For reinforcement learning trained models, as expected by our hypothesis, increases in compression degree reduced both performance and resource utilization. Within naively compressed models, we observed positive correlations between CPU/time resource utilization and performance. Fine-tuning of compressed models with compression built into the training loop proved to be an effective way to recover good performance with compressed models without increasing inference resource usage, especially as compared with simple matrix shrinking.

5 CONTRIBUTION

There was a lot of collaboration on all aspects of the project, and Kunal was responsible primarily for implementing quantization, Nidhir focused on integrating low-rank approximation and weight pruning techniques, Andrew prioritized developing the measurement utilities, and Gaurav handled environment setup, management, and data analysis.

REFERENCES

[1] García-Ramírez, J., Morales, E.F., Escalante, H.J. (2022). "Model Compression for Deep Reinforcement Learning Through Mutual Information." In: Bicharra Garcia, A.C., Ferro, M., Rodríguez Ribón, J.C. (eds) Advances in Artificial Intelligence – IBERAMIA 2022. IBERAMIA 2022. Lecture Notes in Computer Science(), vol 13788. Springer, Cham. https://doi.org/10.1007/978-3-031-22419-5_17.

[2] Gysel, Philipp, Jon Pimentel, Mohammad Motamedi, and Soheil Ghiasi. "Ristretto: A Framework for Empirical Study of Resource-Efficient Inference in Convolutional Neural Networks." *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 11, November 2018, pp. 5784–89. https://doi.org/10.1109/TNNLS.2018.2808319.

[3] Krishnamoorthi, Raghuraman. "Quantizing Deep Convolutional Networks for Efficient Inference: A Whitepaper." *arXiv*, June 21, 2018. http://arxiv.org/abs/1806.08342.

[4] Krishnan, Srivatsan, Maximilian Lam, Sharad Chitlangia, Zishen Wan, Gabriel Barth-Maron, Aleksandra Faust, and Vijay Janapa Reddi. "QuaRL: Quantization for Fast and Environmentally Sustainable Reinforcement Learning." *arXiv*, November 13, 2022. http://arxiv.org/abs/1910.01055.

[5] Lin, Darryl, Sachin Talathi, and Sreekanth Annapureddy. "Fixed Point Quantization of Deep Convolutional Networks." In *Proceedings of The 33rd International Conference on Machine Learning*, 2016, pp. 2849–58. https://proceedings.mlr.press/v48/linb16.html.

[6] Neill, James O'. "An Overview of Neural Network Compression." *arXiv*, August 1, 2020. http://arxiv.org/abs/2006.03669.

[7] "Quantization for Fast and Environmentally Sustainable Reinforcement Learning," September 27, 2022. https://blog.research.google/2022/09/quantization-for-fast-and.html.

[8] Yao, Zhewei, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. "ZeroQuant: Efficient and Affordable Post-Training Quantization for Large-Scale Transformers." *arXiv*, June 3, 2022. https://doi.org/10.48550/arXiv.2206.01861.

[9] Yao, Zhewei, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, et al. "HAWQV3: Dyadic Neural Network Quantization." *arXiv*, June 23, 2021. https://doi.org/10.48550/arXiv.2011.10680.

[10] Yazdanbakhsh, Amir, Ahmed T Elthakeb, Prannoy Pilligundla, and FatemehSadat Mireshghallah Hadi Esmaeilzadeh. "ReLeQ: An Automatic Reinforcement Learning Approach for Deep Quantization of Neural Networks."

[11] Zhou, Xiao, Weizhong Zhang, Hang Xu, and Tong Zhang. "Effective Sparsification of Neural Networks with Global Sparsity Constraint." In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 3598–3607. https://doi.org/10.1109/CVPR46437.2021.00360.

In this section, we present a selection of additional graphs that were generated during our extensive analysis aimed at uncovering correlations between various compression techniques and performance benchmarks. It is worth noting that there exist intriguing patterns within this dataset that warrant further exploration.

6 APPENDIX

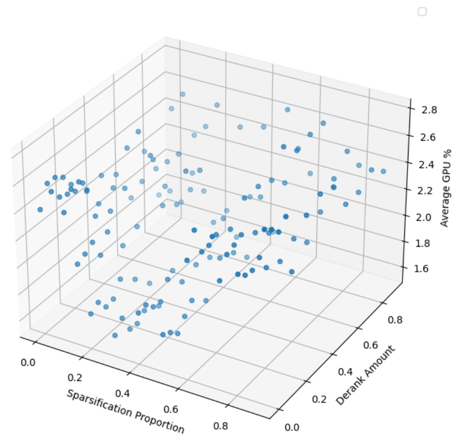

Figure 13

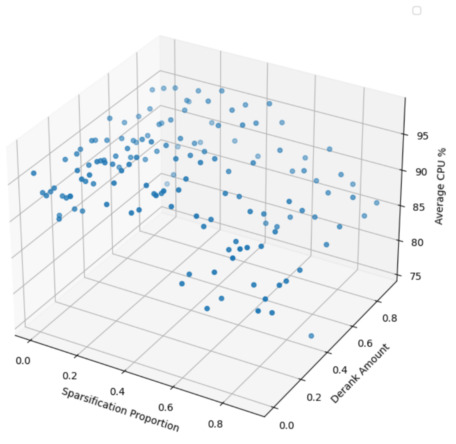

Figure 16

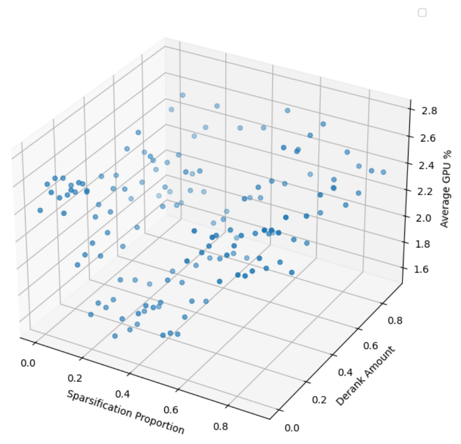

Figure 14

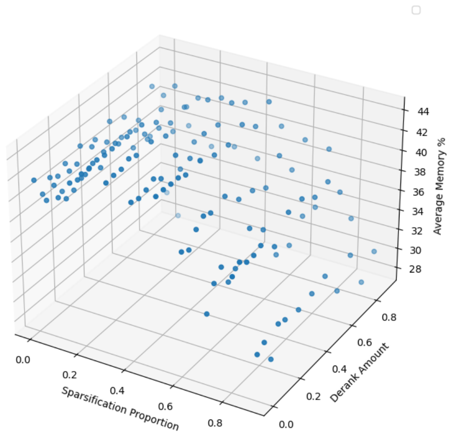

Figure 17

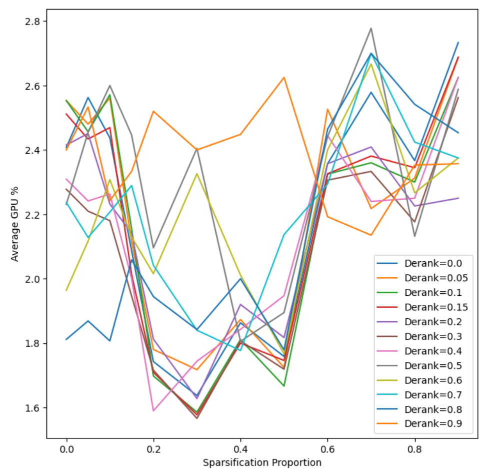

Figure 15

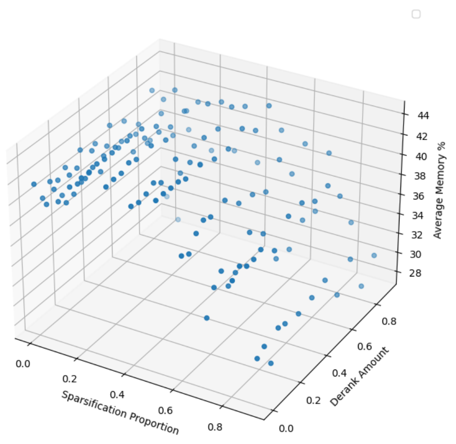

Figure 18

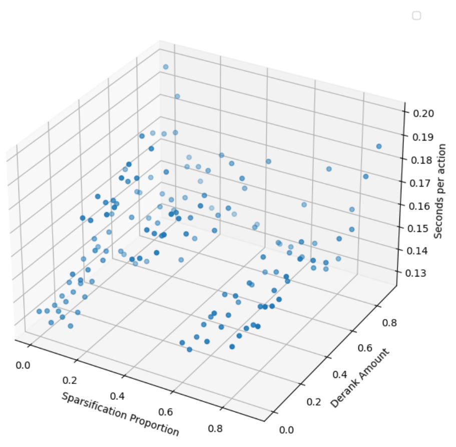

Figure 19

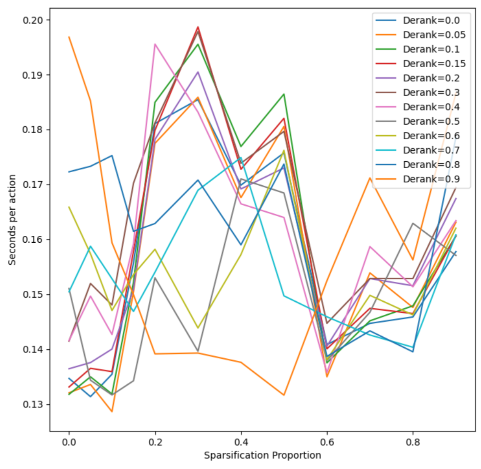

Figure 21

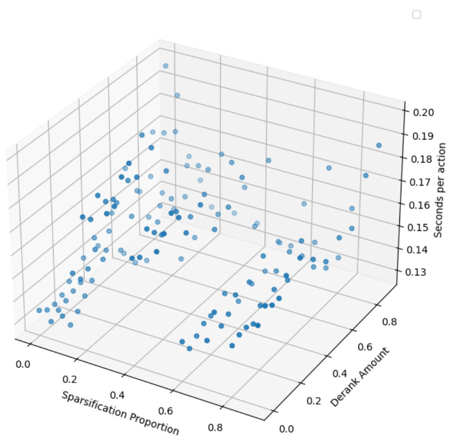

Figure 20