Pied Piper: ZFP Compression

HyunJun Park
Computer Science
University of California, Irvine
Irvine, CA
hyunjup4@uci.edu

Alexander Grindrod
Computer Science and Engineering
University of California, Irvine
Irvine, CA
agrindro@uci.edu

Rohan Gupta
Computer Science
University of California, Irvine
Irvine, CA
rohang5@uci.edu

Varun Nawathey
Computer Science
University of California, Irvine
Irvine, CA
vnawathe@uci.edu

***Abstract*—This paper will discuss the use of ZFP compression in various methods to reduce the size of model parameters and the overall checkpoint file size.**

I. Introduction

Error Bounded Lossy Compression (EBLC) refers to any compression algorithm that allows the user to limit/control the aggressiveness of lossy compression.

ZFP is an EBLC algorithm with parameters such as Tolerance and Precision to enforce control over data loss. The project aims to reveal how well ZFP performs in compressing the weights of various ML model architectures ranging from VGG, ResNet, and Bert.

The project also compares/evaluates the use of ZFP compression against/with similar existing techniques like Quantization, Pruning, and overall lossless compression (zlib). Most importantly, we aim to see if ZFP is a viable compression algorithm for reducing model size while minimizing the loss of performance.

A. Motivation

Pre-trained models, such as large language models (LLMs) and large scale convolutional neural networks (CNNs), have extensive storage requirements. Efficient compression methods like ZFP could address this issue without severely impacting model accuracy. This project explores the potential of ZFP compression as an alternative to traditional quantization and as a supplement to pruning and clustering.

B. Problem Statement

The primary objective is to evaluate the effectiveness of ZFP compression on model weights, as many popular models are substantial in size. With the rise in popularity of LLMs, the issue of model size is significantly more prevalent. ZFP can offer a more flexible and efficient compression approach, particularly for models where quantization may sacrifice too much data.

II. Background and Prior Work

While traditional lossless compression methods, such as ZIP and Huffman Coding, are being wildly used, it is hard to expect a huge compression effect for those lossless algorithms. ZFP was developed as a method for compressed floating-point arrays, primarily used in scientific computing. Previous work has been effective in reducing data size with a controlled loss of precision. One of which explored the application of ZFP compression to training data [1]. Our project builds on this foundation by applying ZFP to the weights and biases of various deep-learning models and comparing the results with other compression techniques.

III. Approach

A. Applying ZFP Compression

We used the ZFP implementation provided by LLNL [2]. The algorithm has been analyzed and verified [3]. Generally ZFP compression is not used in favor of quantization and other lossless algorithms, so it would be interesting to see the results. The ZFP compression has been applied to several benchmark models:

- **Small LLM (DistilBERT)**: Apply ZFP to the model weights (both un-quantized and quantized) using various tolerance values between 2^{-27} and 2^{2}.
- **ResNet-18**: Apply ZFP to a fine-tuned ResNet-18 model provided by PyTorch on the CIFAR-10 dataset, testing various tolerance and precision values.
- **VGG-11**: Apply ZFP to VGG-11, analyzing the effects of varying tolerance, and varying the sparsity of the model weights before compressing.
- **VGG-19**: Compared ZFP compression to aggressive quantization and clustering.

B. Comparative Analysis

The compression ratios of ZFP were compared against lossless compression (zlib), FP16 quantization, and no compression. The project also explores the use of quantization, pruning, and weight clustering followed by ZFP to determine

if it offers any further reduction in model size. Different architectures were used to ensure ZFP could be used universally.

IV. Analysis

A. DistilBERT: ZFP Compression Performance

Pre-trained models, like the Large Langauge Model (LLM), are large. In order to save storage on the device, storage level compression is necessary. The ZFP compression algorithm has been applied to the DistilBERT model's weights with various tolerance values to elaborate the relationship between the model size and the value of the tolerance parameter. While tolerance values between 2^{-27} and 2^{2} have been tested, the DistilBERT model's accuracy and compression ratio also has been logged to provide a more profound analysis.

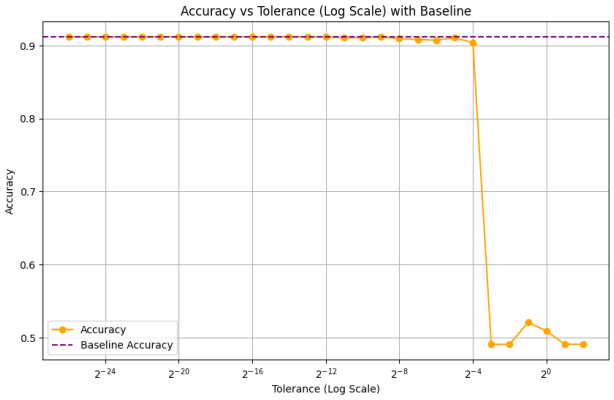

Fig. 1. Accuracy for various Tolerance values

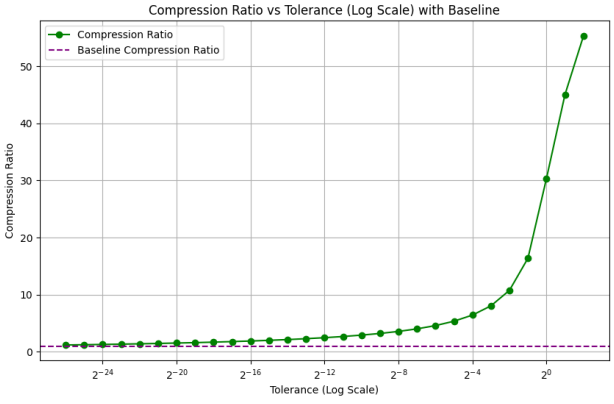

Fig. 2. Compression Ratio for various Tolerance values

The experiments suggest that increasing the tolerance value of the ZFP algorithm yields a higher compression ratio; however, the accuracy plot shows significant drops around $2^{-3}(= 0.125)$. It is notable that we were able to achieve a huge compression ratio of 7 while maintaining the baseline model's accuracy.

The main reason for the accuracy drop could be found by comparing decompressed with original values.

Tolerance	Decompressed Value
Original	-0.016623475
0.001	-0.016658783
0.124	-0.017089844
0.125	**-0.005859375**

The table given above suggests that when the tolerance value becomes larger than 2^{-3} (or 0.125), it loses a significant amount of information.

B. DistilBERT: ZFP Compression with Quantization

The ZFP is also known for being good at compressing redundant data, such as a large group of the same bits. In this section, the ZFP compression algorithm has been applied to the DistilBERT model's quantized weights with various tolerance values to elaborate the relationship between the model size and the value of the tolerance parameter with quantized weights, expecting yielding a higher compression ratio due to more redundant bits in quantized weights. The same tolerance values ($2^{-27} \sim 2^{2}$) have been used with bfloat16 and float8_e5m2 quantized model's weight.

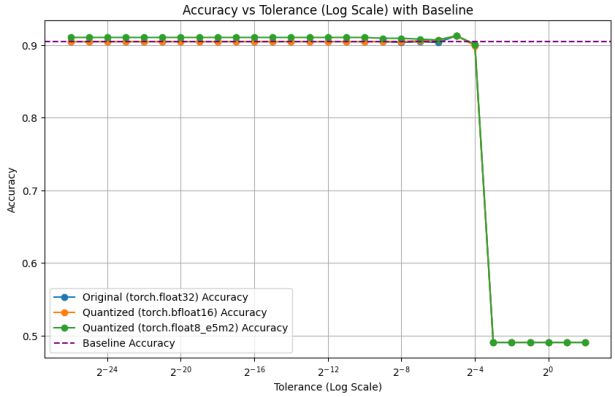

Fig. 3. Accuracy for various Tolerance values (with Quantization)

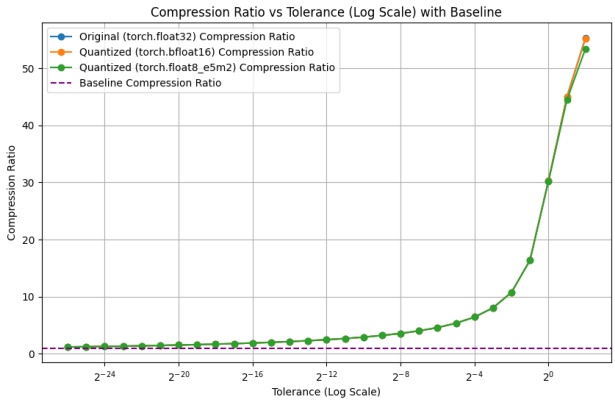

Fig. 4. Compression Ratio for various Tolerance values (with Quantization)

Interestingly, the resulting figure suggests that there is no huge difference between applying quantization to the weight before applying ZFP compression.

Methods	Value
Original	-0.01662347465
Quantization (float8_e5m2)	-0.01562500000
ZFP Decompression	-0.01171875000

The table given above suggests that the ZFP algorithm itself applies some level of quantization. In this case, there is no big difference between applying ZFP and applying 8-bit quantization, which is why it couldn't have more dramatic compression effects.

C. ResNet-18: Varying Tolerance and Precision

Tolerance and Precision can be varied in order to control the extent of lossy compression applied to the model weights. After sweeping through a range of values, tolerance seemed to work best at 2^{-4} error. Refer to Fig 5.

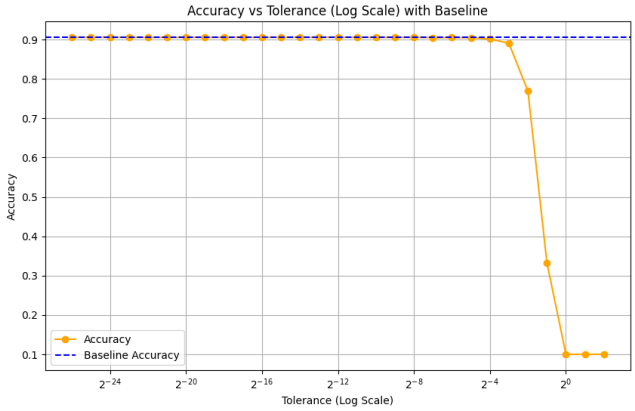

Fig. 5. Accuracy for various tolerance values

Meanwhile, precision seemed to work best at 12 bits. Refer to Fig 6.

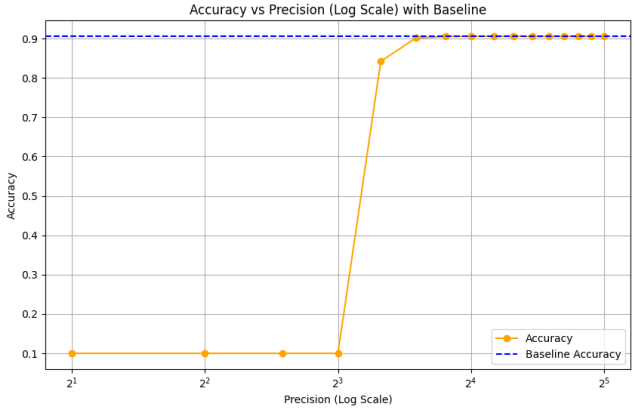

Fig. 6. Accuracy for various precision values

Precision performs similarly to quantization in that both can control how many bits represent the data; however precision has more of a fine-grained control. However, tolerance can be a better alternative as its error value can provide an even finer range of control. ZFP would be ideal in compressing weights if quantization takes too much information away, but the base model size is also still too large.

D. ResNet-18: Comparison against zlib and FP16

The use of ZFP compression vastly outperforms lossless zlib compression with minimal loss to accuracy if any. It also shows better performance against FP16 by being able to compress more data while incurring a cost to accuracy of $< 1\%$. Refer to Fig 7.

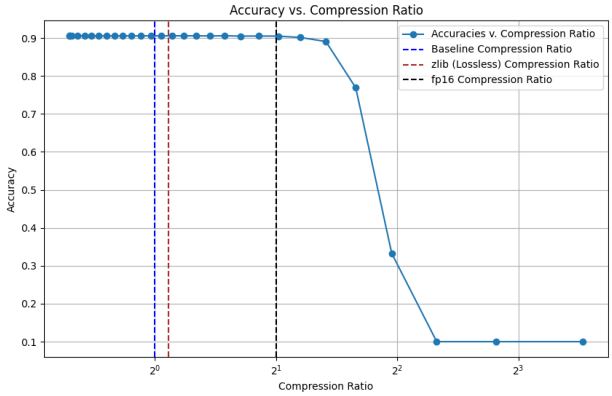

Fig. 7. Accuracy at various compression ratios

E. Comparison with Quantization

ZFP provided better control over model size compared to traditional quantization methods. However, combining quantization with ZFP did not yield as significant improvements as anticipated, suggesting that ZFP alone is quite effective.

F. VGG-11: ZFP Compression with various Tolerance Values

The ZFP compression algorithm was applied to PyTorch's pre-trained VGG-11 model with various levels of Tolerance, ranging from 1 to 2^{-27}. The model size was derived from storing all of the model's named parameters that had a gradients in a pickle (pkl) file and getting the size. The compression ratio and model size plotted against Tolerance can be seen in Figure 8. and Figure 9. This demonstrates the ability of ZFP to compress the VGG-11 model up to a remarkable factor of more than 160 with Tolerance 1. One important thing to notice is that the compressed model size decreases at a relatively linear rate (from tolerance 2^{-27} to 2^{-4}) while the Tolerance increases exponentially. While this demonstrates the intense compression ability of ZFP, we must compare this to the permanence of the accuracy.

Figure 10. and 11. below demonstrate the permanence of ZFP after decompressing the model with various levels of Tolerance. As we can see in the Decompressed Accuracy vs. Tolerance graph, the decompressed model is able to maintain the original accuracy of 57.78% from tolerance level 0 to 2^{-6}, but then drops considerably. While this means we can't obtain

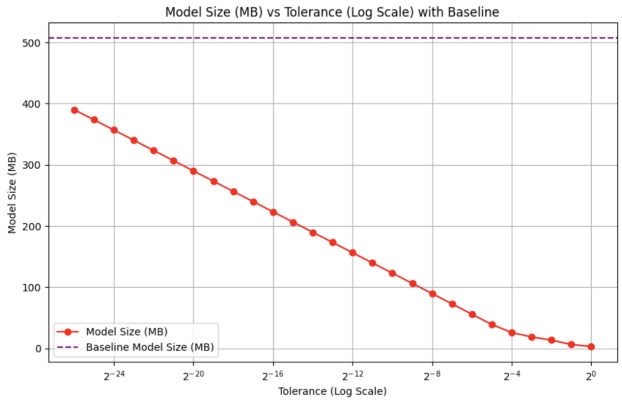

Fig. 8. Compressed Model Sizes vs. Tolerance

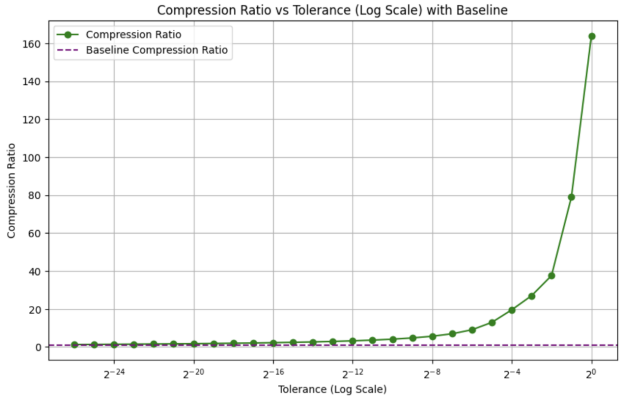

Fig. 9. Compression Ratios vs. Tolerance

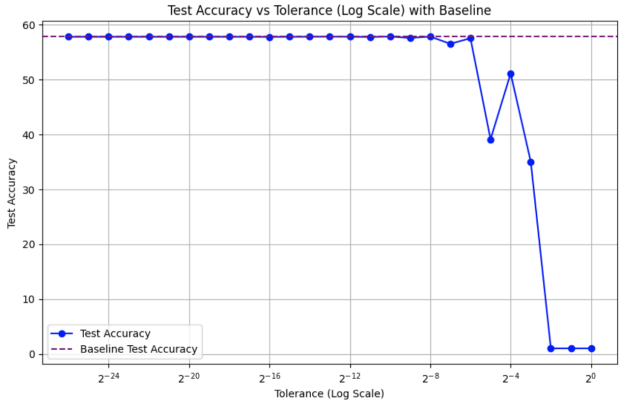

Fig. 10. Decompressed Accuracy vs. Tolerance

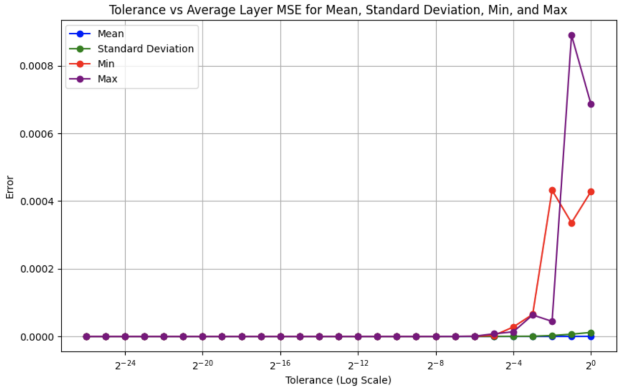

Fig. 11. Decompressed Model Weight MSE vs. Tolerance

the 160x model size compression without accompanying it with an accuracy of near 0, we can still get a 10x compression rate while maintaining accuracy with Tolerance level 2^{-6}. Figure 11. demonstrates the Average Mean Squared Error (MSE) for each Layer in the model after decompressing. This figure demonstrates the powerful ability of ZFP to maintain the Mean and Standard Deviation of the model weights incredibly well, but performs worse with values further away such as the Maximum and Minimum with tolerance levels of 2^{-4} or greater.

G. VGG-11: ZFP Compression with various Sparsity Levels

This section covers the affects of applying ZFP to models with various levels of sparsity (0-100%) with a fixed Tolerance level of 0.0002 (2^{-12}). This Tolerance level was chosen because as shown in the previous graph, it has a minimal effect on the model weights with respect to MSE and accuracy, so we can focus on the effects of sparsity by itself. Figure 12. demonstrates the change in Compression Rate vs. Sparsity, illustrating compression ratios of up to 9x when reaching a Sparsity of 1. Figure 13. conveys the model size of the compressed model after compressing, and the ratio is represented by Fig 12.

Figure 14. illustrates the degradation of accuracy after compressing and decompressing a pruned model. We can see

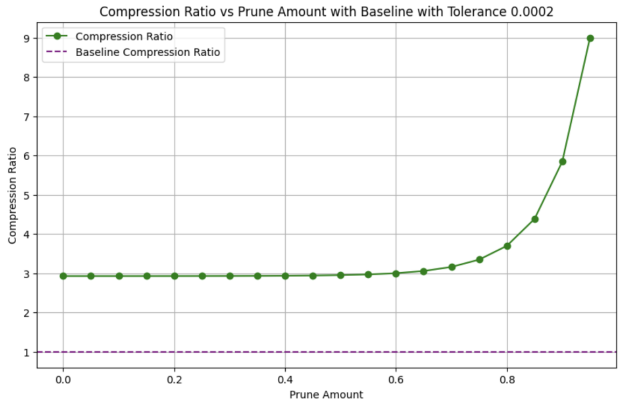

Fig. 12. Compression Rate vs. Sparsity

that at about a sparsity level of 0.6, accuracy begins to drop extremely. Combined with the Compression Ratio graph, we can see that we can get a compression of about 3x while still maintaining original accuracy. Figure 15. demonstrates the Total MSE for Model Weights after decompression, and it agrees with Figure 11. in that ZFP does a good job maintaining mean and standard deviation while doing worse with further values such as the maximum and minimum.

Surprisingly, Figure 16. demonstrates a key examination of

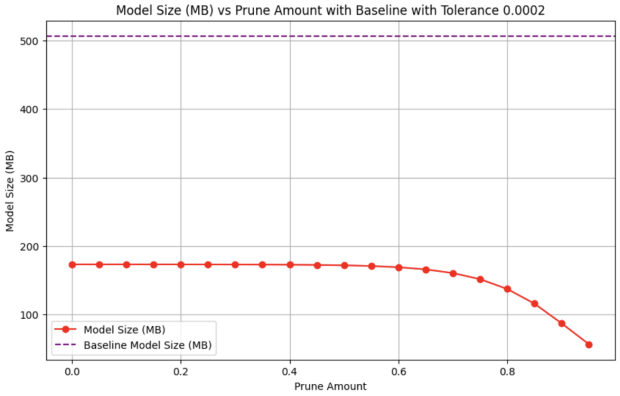

Fig. 13. Model Size vs. Sparsity

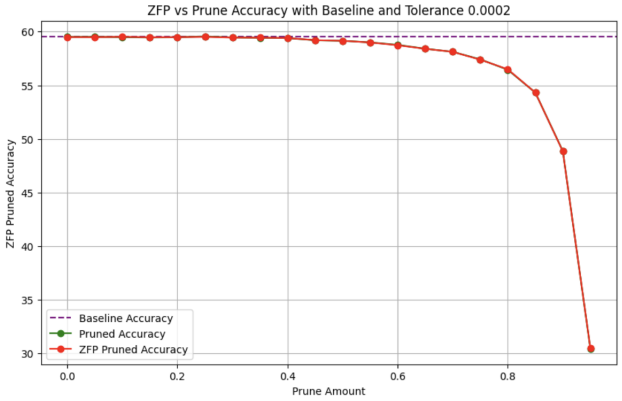

Fig. 14. Accuracy vs. Sparsity

the Sparsity after decompressing, which is that ZFP does not do a good job of keeping model weights with values 0 at exactly 0. Most likely, it decompressed them to something very close to zero, but this has not been verified. A good next step would be to re-prune the model after decompressing to get the original sparsity and seeing whether the model is still able to maintain accuracy.

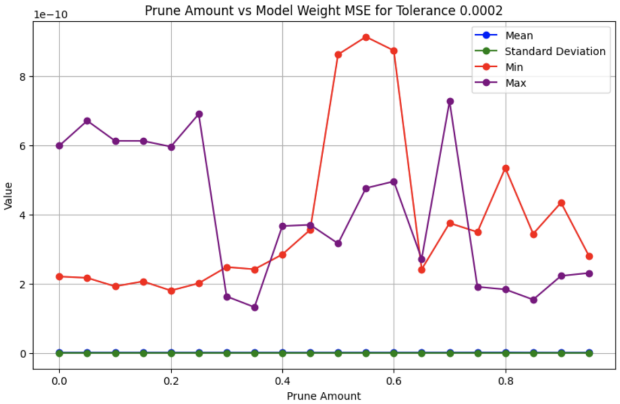

Fig. 15. Decompressed Model Weight MSE vs. Sparsity

Overall, ZFP is a very useful tool for VGG-11 and we can most definitely alter the Tolerance of ZFP and Sparsity of the

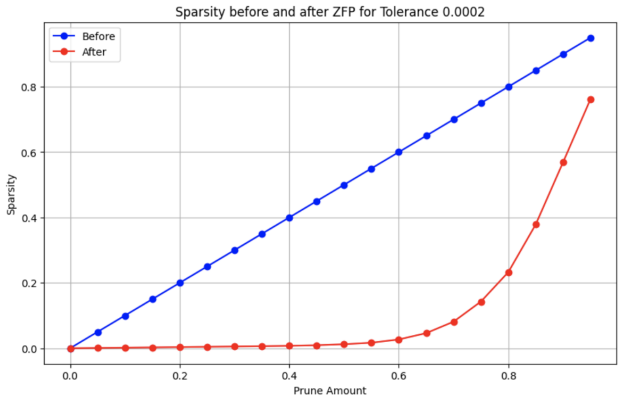

Fig. 16. Sparsity Permanence vs. Sparsity

original model to see substantial benefits in model size while maintaining accuracy.

H. VGG-19: Clustering and quantization: Exploiting Weight Distributions

Quantization is really a form of clustering. Both are piecewise staircase functions over the real value number line. All floating point types are themselves clusterings since they are a discretization of real values: of course, in the case of floating point types, the function is undefined past the maximum and minimum float. Conveniently, many neural network layers have weights that are normally distributed about zero. The variance tends to be small too. Observing these distributions, you'll see that space is wasted since the maximum and minimum elements fall short of the respective maximum and minimums of FP32. Clustering methods let you represent a group of numbers with less distinct values. To lossily represent each weight in a layer, I chose 16 numbers about the approximate mean zero, and replaced each weight with the number closest to it. When it comes time to save the model or transmit it across some channel, you can send over a codebook of the cluster centers and replace each weight with an index into this codebook. For VGG-19, I clustered weights aggressively with 16 centers. Performing this on just the weights matrices that were normally distributed shrunk a roughly 574 MB model to roughly 85 MB. However, this led to a drop in the accuracy of 3%. This was likely due to choosing a single clustering of 16 centers for all weights instead of a distinct clustering per weights tensor. I believe fixing this oversight should bridge the accuracy drop. Since the model is still uncompressed at 85 MB. Further compression can be done on the indices.

V. Methodology

A. Task List

The project involved several key steps:

- Applying ZFP to the weights of various benchmark models and observing the results.
  - This was done across all of the four architectures

  - The weights were saved to a pickle (.pkl) file, and the size was then measured from the file.
- Apply ZFP by varying tolerance and precision and evaluate performance.
  - This was evaluated against zlib lossless compression and FP16 quantization
- Investigating the effects of applying quantization prior to ZFP.
- Investigating the effects of applying pruning prior to applying ZFP.
  - Evaluated the model size compression ratio, MSE of model weights, and persistence of sparsity
- Investigating the effects of tailored quantization prior to ZFP
- Uploading related code and data to our GitHub repository (https://github.com/alex-grindrod/eblc-model-compression).

B. Challenges

One of the primary challenges was understanding the ZFP algorithm and taking the necessary steps to make it fit with our tests. Another challenge was balancing the trade-off between compression ratio and model accuracy.

VI. Ownership

- **HyunJun Park:** Applied ZFP to DistilBERT. Observed effects of Quantization with ZFP
- **Alexander Grindrod:** Applied ZFP to ResNet-18. Observed effects of varying Tolerance and Precision with ZFP
- **Varun Nawathey:** Clustered VGG-19 weights
- **Rohan Gupta:** Applied ZFP to VGG-11. Observed the effects of varying Tolerance and Sparsity when applying ZFP

VII. Conclusion

Our experiments demonstrated that ZFP is an effective compression method for reducing the storage size of ML models while maintaining acceptable accuracy levels. It offers a flexible alternative to traditional quantization methods, especially for models that can tolerate some degree of lossy compression. Future work could explore further optimizations and applications of ZFP in other areas of ML.

Acknowledgment

Thanks to the Lawrence Livermore National Laboratory for providing the zfpy module and supporting documentation.

References

[1] R. Underwood, J. C. Calhoun, S. Di, and F. Cappello, "Understanding the effectiveness of lossy compression in machine learning training sets" https://arxiv.org/html/2403.15953v1#S1.

[2] P. Lindstrom, "Fixed-Rate Compressed Floating-Point Arrays," IEEE Transactions on Visualization and Computer Graphics, vol. 20, no. 12, pp. 2674–2683, Dec. 2014.

[3] J. Diffenderfer, A. Fox, J. Hittinger, G. Sanders, and P. Lindstrom, "Error Analysis of ZFP Compression for Floating-Point Data," SIAM Journal on Scientific Computing, vol. 41, no. 3, pp. A1867–A1898, June 2019.