An Efficient Checkpointing System for Large Machine Learning Model Training

Wubiao Xu
Nanchang Hangkong University
xwbdmx1736@gmail.com

Xin Huang
Kobe University
xin.huang@a.riken.jp

Shiman Meng, Weiping Zhang
Nanchang Hangkong University
{*smeng, wzhang*}*@nchu.edu.cn*

Luanzheng Guo
Pacific Northwest National Laboratory
lenny.guo@pnnl.gov

Kento Sato
R-CCS, RIKEN
kento.sato@riken.jp

***Abstract*—Checkpointing is one of the fundamental techniques to resume training while system fails. It has been generally used in various domains, such as high-performance computing (HPC) and machine learning (ML). However, as machine learning models increase in size and complexity rapidly, the cost of checkpointing in ML training became a bottleneck in storage and performance (time). For example, the latest GPT-4 model has massive parameters at the scale of 1.76 trillion. It is highly time and storage consuming to frequently writes the model to checkpoints with more than 1 trillion floating point values to storage. This work aims to understand and attempt to mitigate this problem. First, we characterize the checkpointing interface in a collection of representative large machine learning/language models with respect to storage consumption and performance overhead. Second, we propose the two optimizations: i) A periodic cleaning strategy that periodically cleans up outdated checkpoints to reduce the storage burden; ii) A data staging optimization that coordinates checkpoints between local and shared file systems for performance improvement. The experimental results with GPT-2 variants show that, overall the proposed optimizations significantly reduce the storage consumption to a constant while improves performance by average 2.1× for checkpointing in GPT-2 training.**

I. Introduction

Machine learning has achieved remarkable success in a variety of fields, such as self-driving cars [1], LLMs [2], image analysis [3], speech and object recognition [4]. Machine learning training has been a long running workload that can last for a few weeks or even months depending on the hardware platform in use. For example, AlphaFold2, a large machine learning model for protein structure prediction, can take about 11 days to train on 128 Google TPUv3 nodes with more than 100TB datasets [5]. On the other hand, faults and failures are found more frequently at those systems as they have been exaggeratedly used by computation- and data-intensive training workloads like crypto mining machines [6]. Checkpointing has been a key component of training workloads to guarantee that, in a system failure, the training can restart from checkpoints rather than from scratch [7].

However, as large machine learning models grow at a rapid pace in size (see Table I), checkpointing a large machine learning model with millions or billions of parameters frequently became a burden to storage and bottleneck to performance.

TABLE I: Large language model examples.

Model	Model size	Checkpoint size
GPT2-large [9]	774M	2.9GB
GPT2-xl [9]	1.5B	5.9GB
Vicuna [10]	7B	13.4GB
OpenOrca [11]	7B	14.5GB
LLama-2 [12]	70B	140GB
GLM [13]	130B	70GB
BLOOM [14]	176B	329GB
GPT3 [15]	175B	700GB

Table I provides a few examples of large language model, including the model size (number of parameters) and storage consumption to write the model to checkpoints. For example, the GPT-3 model has 175 billion parameters; checkpointing the whole GPT-3 model to storage takes 700GB when each parameter takes four bytes. Large language models are still increasingly growing in size. For example, the latest GPT-4 model has 1.76 trillion parameters [8], which is 10× larger than GPT-3.

There have been a few works trying to solve this problem for large model training. For example, Nicola et al. [16] propose Deepfreeze, a checkpoint technique that introduces an asynchronous technique to hide serialization and I/O overhead among all participating processes for better I/O perforamnce. Moha et al. build CheckFreq [17], a method that introduces two-stage checkpoint scheme to avoid GPU stagnation and reduce checkpoint overhead. Guo et al.used PARIS [18], a machine learning method for predicting application resilience, to speed up the recovery of errors. However, they focus on I/O optimizations while fail to take into account the training workload as well as the supermassive data written to storage.

To reduce the storage burden for checkpointing in large model training, we propose a periodic cleaning strategy, which significantly reduce the storage consumption to a constant, and a data staging optimization scheme that improves the checkpointing performance by 2.1× on average. Our contributions are as follows.

1) We characterize the checkpointing with respect to storage and performance in large model training with representa-

979-8-3503-5554-3/24/$31.00 ©2024 IEEE
DOI 10.1109/SCW63240.2024.00127

tive large language models.
2) We introduce a periodic cleaning strategy, by deleting old checkpoints periodically, which keeps the checkpointing storage consumption to a constant.
3) We propose a data staging optimization that coordinates checkpoints between local and shared file systems.
4) Evaluation on GPT-2 variants that includes comparison against the original checkpointing strategy.

II. Background

We will first introduce the concept of introducing checkpoints in deep learning, and explain the common methods of checkpoint optimization, its use in deep learning, and the situation in training model restart.

A. Checkpoint application

Checkpoints are widely used in deep learning projects. First, they are used to implement breakpoint recovery in the training process. This means that if the training process is interrupted in the middle, such as due to hardware failure or other reasons, the progress of the training process can be resumed by loading the most recent checkpoint, without the need to start over [19]. This not only saves valuable time, but also reduces waste of resources. Secondly, checkpoints play an important role in transfer learning. Transfer learning is a method of using knowledge of already trained models to accelerate the training of new tasks. Checkpoints preserve the knowledge and parameters of the model, allowing it to be transferred and reused between different tasks. This knowledge transfer can speed up the training process for new tasks because the model already has a certain knowledge base.

B. Existing checkpoint optimization

In order to solve the problem of storage and computing resource consumption caused by checkpoint, researchers have proposed a variety of checkpoint optimization strategies. Among them, model compression is a common strategy to reduce storage requirements by reducing model size [20] [21]. Model pruning is another strategy to reduce the computational complexity of the model by cutting unnecessary parameters [22]. In addition, the quantization method reduces the size of the model by reducing the precision of the weight parameters [23], thus improving the efficiency of the model. These strategies can reduce the size of checkpoints without affecting the performance of the model and improve the efficiency of resource utilization.

Despite the existence of multiple checkpoint optimization strategies, there are still some challenges in practical application. For example, how to effectively crop checkpoints to reduce model size while maintaining model performance is still an open question. This study aims to solve this problem, and proposes a checkpoint clipping method based on network layer combination, combined with dynamic storage strategy, to reduce the size and speed of checkpoint storage, while ensuring the accuracy of restart.

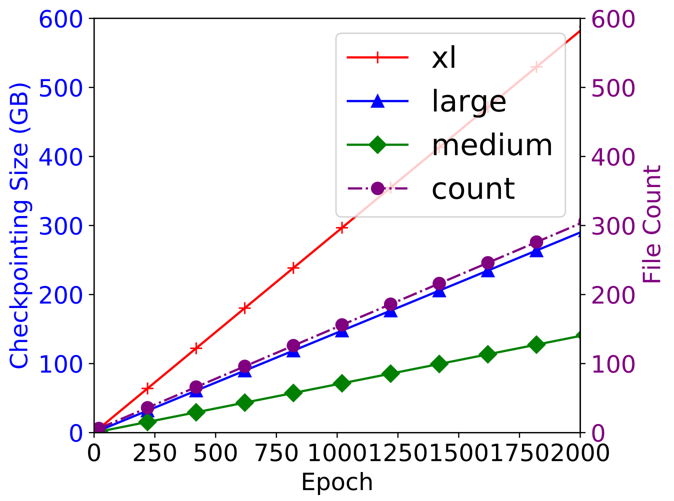

Fig. 1: Original: the checkpointing size and number of checkpoint files at each epoch by GPT2 variants in training.

III. Problem and Optimization

The current checkpointing technique in machine learning training stores all the model parameters (the whole model) to checkpoints periodically. Moreover, the outdated checkpoint files are not removed, which would cause storage overflow as the training proceeds. We characterize the problem with representative GPT workloads and propose two optimizations for performance improvement.

A. Problem definition/description

As the large model training proceeds, we observe an increasing growth in the number of accumulated checkpoint files and storage consumption for checkpointing (see Figure 1). The checkpointing latency slowly increases for later checkpointing as the storage is taken progressively (see Figure 3). We also note that the time spent on writing checkpoints became a performance bottleneck depending on checkpointing frequency. For example, in the experiment we designed, for the GPT2-xl model, the epoch time of a single training is about 1s, but the time to save a checkpoint is more than 15x of the training time, the whole experiment stage is displayed for one hour, but the cost of saving the check tape is as high as 42 %. Although this is because checkpoints are designed to be frequent, it still shows that checkpoints account for a significant portion of the time.

B. Periodic cleaning strategy

The current checkpointing technique retains all checkpoints including outdated in model training, resulting in continuous growth of storage footprint for checkpointing. As the training progresses, the significance of new checkpoints relative to old ones gradually increases. Moreover, for the purpose of resilience only the latest checkpoints are required for restarting the training upon a failure. As inspired, we propose a periodic cleaning strategy. By regularly removing unnecessary

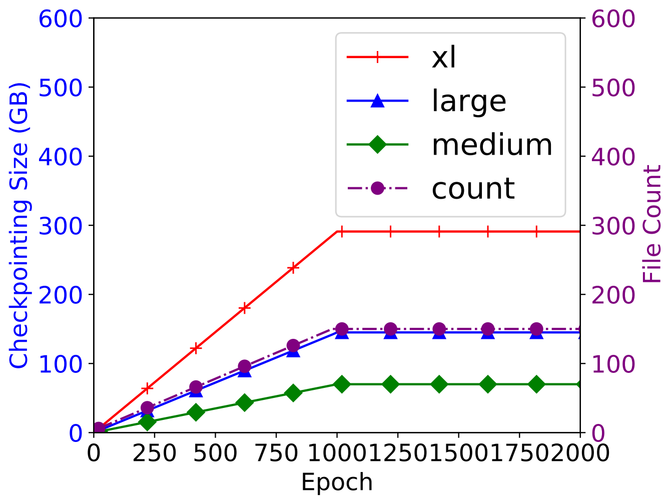

Fig. 2: Periodic cleaning strategy: the checkpointing size and number of checkopint files at each epoch by GPT2 variants.

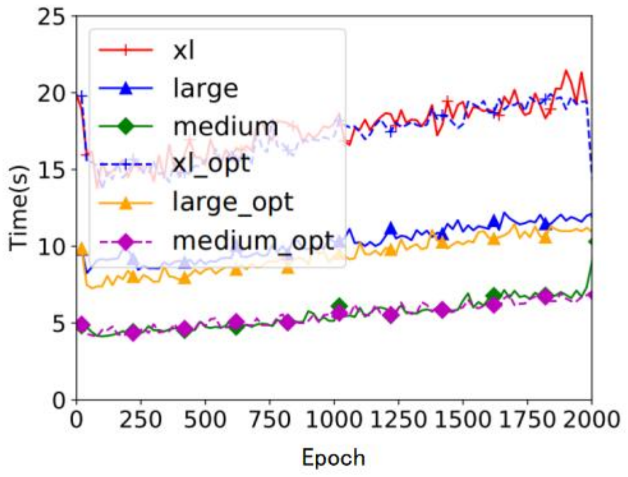

Fig. 3: Periodic cleaning strategy (opt) vs. original: checkpointing time at each epoch by GPT2 variants.

outdated files *asynchronously*, it ensures that only the latest model parameters are retained in storage. This significantly reduces storage space requirements and avoids excessive storage footprint.

C. *Data staging optimization*

The default checkpointing technique writes checkpoints to shared file systems directly. Shared file systems are perfect for coordinating data access among many processes across distinct compute nodes. Unfortunately, shared file systems have much slower I/O than node-local file systems, such as ramdisk. Also, shared file systems are accessed by many compute nodes concurrently and are more likely to encounter I/O congestion. Therefore, we propose a staging optimization which first writes checkpoints to local file systems, particularly ramdisk, and then transfers checkpoints to shared file system, asynchronous to the model training. This approach is designed to enable faster and more efficient checkpointing.

IV. Evaluation

Experimental setup. We use the GPT2 model [9] and its variants for evaluation as it is the largest model that can fit to our machine. We limit the training course to three hours for each test we run. We set the checkpointing frequency to the default setting (every 20 epochs).

Platforms. All experiments were run on a Linux server with two AMD EPYC 7763 CPUs, each with 64 cores, and eight A100 GPUs, each with 80 GB of memory and 2.0 TB of memory. Python3.8 is used as the programming language and TensorFlow 2.4.0 are used.

We selected GPT2 model suitable for text generation task for experimental verification.We scaled the experiment time to 3 hours to show the early effect of the experiment, and set the storage checkpoint frequency to store once every 20 epochs to observe the storage time comparison under various methods.

A. *Periodic cleaning*

In this section, we will describe the experimental design and methodology of a periodic cleaning strategy to verify how storage efficiency can be improved by removing early checkpoints. During model training, as the training time goes by, the number of checkpoints used to store the model in training increases, so does the required storage consumption, as shown in the Figure 1. By developing a periodic cleaning strategy, we are able to get the storage consumption under control and retain a constant storage consumption(see Figure 2).

In addition, we observe that, as training proceeds, the time of writing a single checkpoint increases slowly. So regularly cleaning up early checkpoints is an effective way to reduce the burden on the system. By continuously testing different types of models, using optimization methods and collecting the time to store a single checkpoint, when no optimization is performed, the minimum storage checkpoint time for GPT2-xl is 13.7s and the maximum time is 20.2s over the first 2000 epochs observed. For GPT2-large, the minimum time is 7.2s and the maximum time is 11s. For GPT2-medium, the minimum time is 4.1s and the maximum time is 7s. As shown in Figure. 3, periodically cleaning checkpoints can reduce the time required for single storage to a certain extent, reduce the entire training process and the memory requirement of the project.

B. *Staging*

In order to improve the storage speed and reduce the overhead of data transmission, the local storage, such as ramdisk, at a compute node can be used to store checkpoints. Local storage is much faster than shared storage, thereby local storage is more preferred if available. However, the capacity of the local storage at compute nodes is limited.

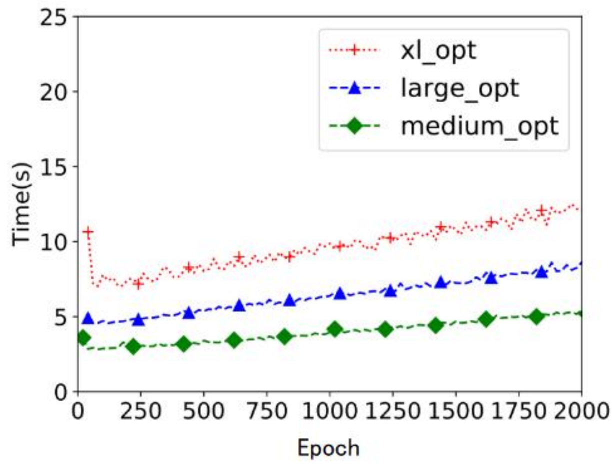

Fig. 4: Staging (opt): checkpointing time at each epoch by GPT2 variants.

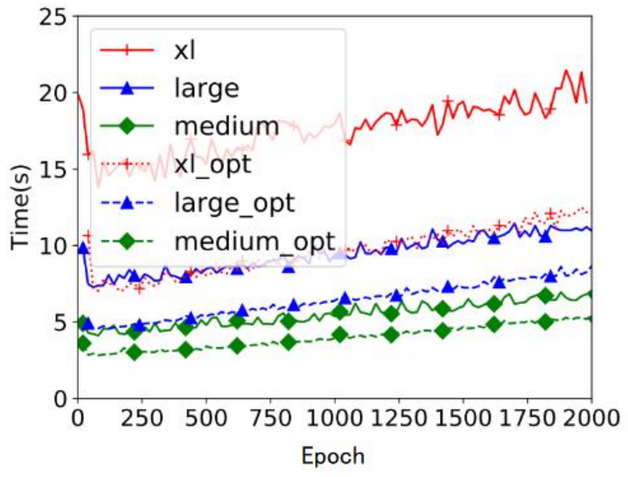

Fig. 5: Putting all together (opt; including periodic cleaning and staging) vs. original: checkpointing time at each epoch by GPT2 variants..

Therefore, it is necessary to coordinate data movement between local and shared storage to avoid storage overflow. We test different storage methods for the same model training process and compare the performance difference between storing checkpoints using compute node vs. storing them using shared storage. In the experiment, the checkpointing time is measured for comparison, demonstrating advantage of the checkpoint staging optimization. Figure 4 shows the time of writing a single checkpoint at each epoch by GPT2 variants with staging optimization. The performance is improved by 1.9× on average compared with the original checkpointing.

C. Putting all together

Finally, we combine the two optimization methods together. As shown in Figure 5, the combined optimization scheme achieves 2.1× speedup on average compared with the original checkpointing. Overall, the checkpointing time of the three GPT models (xl, large, and medium) is improved significantly.

V. Discussion and Future Work

Disscussion In this study, the exploration of dynamic memory strategies and temporary memory storage checkpoints provides valuable insights into the efficiency and adaptability of neural networks. Dynamic memory strategies reduce memory load by dynamically adjusting checkpoint memory usage based on model requirements and changes in computing resources. Temporary memory storage checkpoints can save the state of the model more quickly during training so that training can continue. From the experiments conducted, there are several points that can be discussed: (1) Losing early checkpoints may increase the risk of model training, and later in training, the model may be more difficult to converge, because losing early checkpoints may result in the loss of some useful information. (2) If the compute node's native memory is insufficient to accommodate the entire checkpoint data, memory overflow and training interruption may result.

Future Work In future research on neural network checkpoint storage optimization, these methods can be explored:

1) Exploring intelligent checkpoint management strategies based on dynamic adjustment of the model training process to achieve a more efficient balance between memory utilization and model performance, and consider using reinforcement learning or adaptive algorithms to automatically select which checkpoints to keep, thereby improving the stability of training
2) Investigating more efficient asynchronous memory write algorithm to reduce the time cost of memory write operation, exploring the implementation of faster asynchronous write process in the local memory of computing nodes, and improving the overall training efficiency.
3) Researching on the integration of distributed storage systems and asynchronous transmission to optimize data distribution and synchronization.

VI. RELATED WORK

Checkpoint/Restart. Checkpoint technology has been extensively studied in distributed systems, and checkpoint optimization usually involves reducing memory latency, coordinating multiple checkpoint to achieve reliable restarts. In order to further explore different optimal checkpoint schemes, many studies have made different directions of exploration. An approach called GEMINI aims to achieve fast failure recovery for distributed training through checkpoints in memory [24]. A traditional checkpoint approach might involve writing model parameters to disk, but GEMINI chose to save the checkpoint in memory to reduce the time it takes for the checkpoint to recover. There is also a model algorithm that optimizes the lifetime and memory location of tensors used to train neural networks [25]. The method automatically reduces the memory usage of existing neural networks. This algorithm designs a method that requires only $O(\sqrt{n})$ memory to train an n-layer network, requiring only one additional forward propagation computation in each small batch to exchange the computation for the possibility of memory, providing a more memory-efficient training algorithm.

ML performance optimization. Current machine learning deep learning checkpoint storage optimization can be summarized in two main aspects. One is model parameter optimization, which includes quantizing the weights of floating point numbers to integers of lower numbers [21] [23] [26], removing pruning of unimportant or redundant weights in the model. The other is to optimize the storage and transmission, hierarchical checkpoints, and incremental saving of weight parameters [27], compression algorithms and heterogeneous storage to reduce memory overhead, network transmission optimization to optimize the loading speed [28]. Our optimization solution focuses on some improvements in storage transfer and optimized memory architecture, eliminating useless early checkpoints in order to save storage space, load time, and data overhead. Using temporary memory can improve training efficiency and reduce disk I/O overhead. This has a positive impact on deep learning model training.

VII. Conclusion and future work

This paper presents a new systematic method for preserving training checkpoints for deep learning models. This method maintains sufficient storage and reduces storage load by actively removing early useless checkpoints, and saves training checkpoints with temporary storage coordination, which significantly reduces the time consumption of a single save checkpoint. At the same time, on the premise of protecting the security of checkpoints, the time cost of transferring redundant checkpoints is reduced.

References

[1] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, 2020.

[2] D. Adiwardana, M.-T. Luong, D. R. So, J. Hall, N. Fiedel, R. Thoppilan, Z. Yang, A. Kulshreshtha, G. Nemade, Y. Lu *et al.*, "Towards a human-like open-domain chatbot," *arXiv preprint arXiv:2001.09977*, 2020.

[3] D. Shen, G. Wu, and H.-I. Suk, "Deep learning in medical image analysis," *Annual review of biomedical engineering*, vol. 19, pp. 221–248, 2017.

[4] S. Jain, K. Pulaparthi, and C. Fulara, "Content based image retrieval," *Int. J. Adv. Eng. Glob. Technol*, vol. 3, pp. 1251–1258, 2015.

[5] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko *et al.*, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[6] K. R. Hari, S. Y. Sai *et al.*, "Cryptocurrency mining–transition to cloud," *International Journal of Advanced Computer Science and Applications*, vol. 6, no. 9, 2015.

[7] Y. Liu, R. Nassar, C. Leangsuksun, N. Naksinehaboon, M. Paun, and S. L. Scott, "An optimal checkpoint/restart model for a large scale high performance computing system," in *2008 IEEE international symposium on parallel and distributed processing*. IEEE, 2008, pp. 1–9.

[8] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[9] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[10] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J. E. Gonzalez *et al.*, "Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality," *See https://vicuna. lmsys. org (accessed 14 April 2023)*, 2023.

[11] T. Vergho, J.-F. Godbout, R. Rabbany, and K. Pelrine, "Comparing gpt-4 and open-source language models in misinformation mitigation," *arXiv preprint arXiv:2401.06920*, 2024.

[12] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[13] Z. Du, Y. Qian, X. Liu, M. Ding, J. Qiu, Z. Yang, and J. Tang, "Glm: General language model pretraining with autoregressive blank infilling," *arXiv preprint arXiv:2103.10360*, 2021.

[14] B. Workshop, T. L. Scao, A. Fan, C. Akiki, E. Pavlick, S. Ilić, D. Hesslow, R. Castagné, A. S. Luccioni, F. Yvon *et al.*, "Bloom: A 176b-parameter open-access multilingual language model," *arXiv preprint arXiv:2211.05100*, 2022.

[15] L. Floridi and M. Chiriatti, "Gpt-3: Its nature, scope, limits, and consequences," *Minds and Machines*, vol. 30, pp. 681–694, 2020.

[16] B. Nicolae, J. Li, J. M. Wozniak, G. Bosilca, M. Dorier, and F. Cappello, "Deepfreeze: Towards scalable asynchronous checkpointing of deep learning models," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. IEEE, 2020, pp. 172–181.

[17] J. Mohan, A. Phanishayee, and V. Chidambaram, "{CheckFreq}: Frequent,{Fine-Grained}{DNN} checkpointing," in *19th USENIX Conference on File and Storage Technologies (FAST 21)*, 2021, pp. 203–216.

[18] L. Guo, D. Li, and I. Laguna, "Paris: Predicting application resilience using machine learning," *Journal of Parallel and Distributed Computing*, vol. 152, pp. 111–124, 2021.

[19] A. Mousavi, A. B. Patel, and R. G. Baraniuk, "A deep learning approach to structured signal recovery," in *2015 53rd annual allerton conference on communication, control, and computing (Allerton)*. IEEE, 2015, pp. 1336–1343.

[20] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, vol. 28, 2015.

[21] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[22] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 784–800.

[23] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[24] Z. Wang, Z. Jia, S. Zheng, Z. Zhang, X. Fu, T. E. Ng, and Y. Wang, "Gemini: Fast failure recovery in distributed training with in-memory checkpoints," in *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023, pp. 364–381.

[25] B. Steiner, M. Elhoushi, J. Kahn, and J. Hegarty, "Model: memory optimizations for deep learning," in *International Conference on Machine Learning*. PMLR, 2023, pp. 32 618–32 632.

[26] B. Rokh, A. Azarpeyvand, and A. Khanteymoori, "A comprehensive survey on model quantization for deep neural networks," *arXiv preprint arXiv:2205.07877*, 2022.

[27] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[28] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge ai: On-demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2019.