SvedbergOpen
DISSEMINATION OF KNOWLEDGE

International Journal of Artificial Intelligence and Machine Learning

Publisher's Home Page: https://www.svedbergopen.com/

Research Paper Open Access

Gradient Checkpointing And Recomputation Strategies For Deep Learning On Memory-Limited Devices

V. Sivasankari[1*], B. Gayathri[2], Dr. Lincy Roy[3], Yuldasheva Maftuna Mamurjon kizi[4]

[1*]Assistant Professor, Department of Mathematics, Meenakshi College of Arts and Science, Meenakshi Academy of Higher Education and Research, Chennai, Tamil Nadu, India. E-mail: sivasankariv@maher.ac.in
[2]Assistant Professor, Department of Computer Science, Meenakshi College of Arts and Science, Meenakshi Academy of Higher Education and Research, Chennai, Tamil Nadu, India. E-mail: gayathrib@maher.ac.in
[3]Assistant Professor, Kalinga University, Naya Raipur, Chhattisgarh, India. E-mail: ku.lincyroy@kalingauniversity.ac.in, https://orcid.org/0009-0009-6004-1799
[4]Turan International University, Namangan, Uzbekistan. E-mail: maftunayuldasheva0224@gmail.com, https://orcid.org/0009-0001-3047-0295
*Corresponding author: Email: sivasankariv@maher.ac.in

Abstract
The memory bottleneck in training very deep neural networks on memory-limited hardware (e.g., 8-16 GB VRAM consumer GPUs, mobile NPUs, and embedded accelerators) is mainly attributed to the large size of activation memories, which are required to save intermediate feature maps for backpropagation. Gradient checkpointing is one possible method for addressing this memory issue. In particular, intermediate activations that would have been discarded during the forward pass are deleted and then recomputed on demand during the backward pass. This technique trades extra compute for reduced peak memory. Unfortunately, simple checkpointing methods that recompute every lost activation back to the previous checkpoint result in at least 33% compute overhead. This paper proposes GradRecomp, a framework for optimized gradient checkpointing and recomputation, which uses a Dynamic Programming activation cost model to place checkpoints efficiently to maximize memory savings and optimized partial recomputation paths based on the knowledge of cached intermediate values for optimal performance overhead during recomputation. Our results show that with GradRecomp, save up to 32% of full activation caching peak memory, and the overhead is only 5%, 18%, and 8% with respect to ResNet-152, BERT-Large, and ViT-B/16, respectively, compared to 51% memory and 18% overhead with no checkpointing, and 44% memory and 8% overhead with Checkpoint-2.

Keywords: Gradient Checkpointing, Activation Recomputation, Memory-Efficient Training, Deep Networks, Dynamic Programming, Mobile Training, Memory Optimization.

1. Introduction

The memory requirements for training a deep neural network scale with both the number of layers and the size of intermediate feature maps, which must be kept in GPU memory during the forward pass in order to be readily available to the backpropagation pass. A 152-layer ResNet trained on ImageNet with batch size 64 requires over 24 GB of activation memory, far exceeding typical consumer and edge GPU devices [1][9]. Gradient checkpointing (also known as activation recomputation) provides a principled way to trade compute for memory: a certain fraction of the intermediate feature maps generated during the forward pass are dropped instead of being stored, and then recomputed during backpropagation.

The original gradient checkpointing algorithm by Chen et al. Uses an O(sqrt(N)) memory reduction for a network of sqrt(N) layers by placing a checkpoint every O(sqrt(N)) layers and recomputing each dropped segment using the nearest checkpoints. This method optimally trades memory for compute for uniform networks, but works poorly on networks with uneven layer activation sizes and compute costs. That is, a checkpoint placed every sqrt(N) layers under this strategy uses far too much memory on some layer sizes, and recomputes far too much on others.
Reversible network architectures, such as RevNet, achieve $O(N^2)$ activation memory cost. RevNet residual computations can be reversed to reconstruct any intermediate activation from its outputs. However, mathematical

reversibility forces constraints on which operations may be used, prohibiting typical operations like batch normalization and attention, thus limiting them to a small set of networks [3][10].
Develop GradRecomp to address the limitations of uniform checkpointing and reversible networks. Our framework makes use of dynamic programming to determine the optimal placement of checkpoints and a new technique called "partial recomputation" that allows for reduced recomputation costs below those possible with standard gradient checkpointing, while also incorporating the memory and compute costs for each layer to optimally reduce memory given a fixed recomputation overhead.

2. Related Work

Activation Memory Reduction Techniques

Gradient checkpointing [1] presented the O(sqrt(N)) memory reduction technique for uniform checkpointing. Recomputation to reduce activation memory has been used to train transformers and achieved up to 50-70% activation memory reduction at 20-30% overhead cost on transformers like BERT and GPT-2 [4]. ZipNN explored lossless compression of intermediate activations to reduce memory bandwidth at no extra computational cost, as a complement to recomputation strategies [5].

Reversible and Memory-Efficient Architectures

Reversible residual networks [3] offer $O(N^2)$ memory cost. This architecture has also been applied to transformers (Reformer [6]), achieving $O(N^2)$ to O(N log N) memory usage at a significant compute cost overhead. This works by using attention mechanisms that scale locally (locally sensitive hashing) and reversing residual connections. While this reduces memory, reversibility is restrictive, disallowing complex operations commonly used in many networks, such as standard batch normalization.

Optimal Recomputation Scheduling

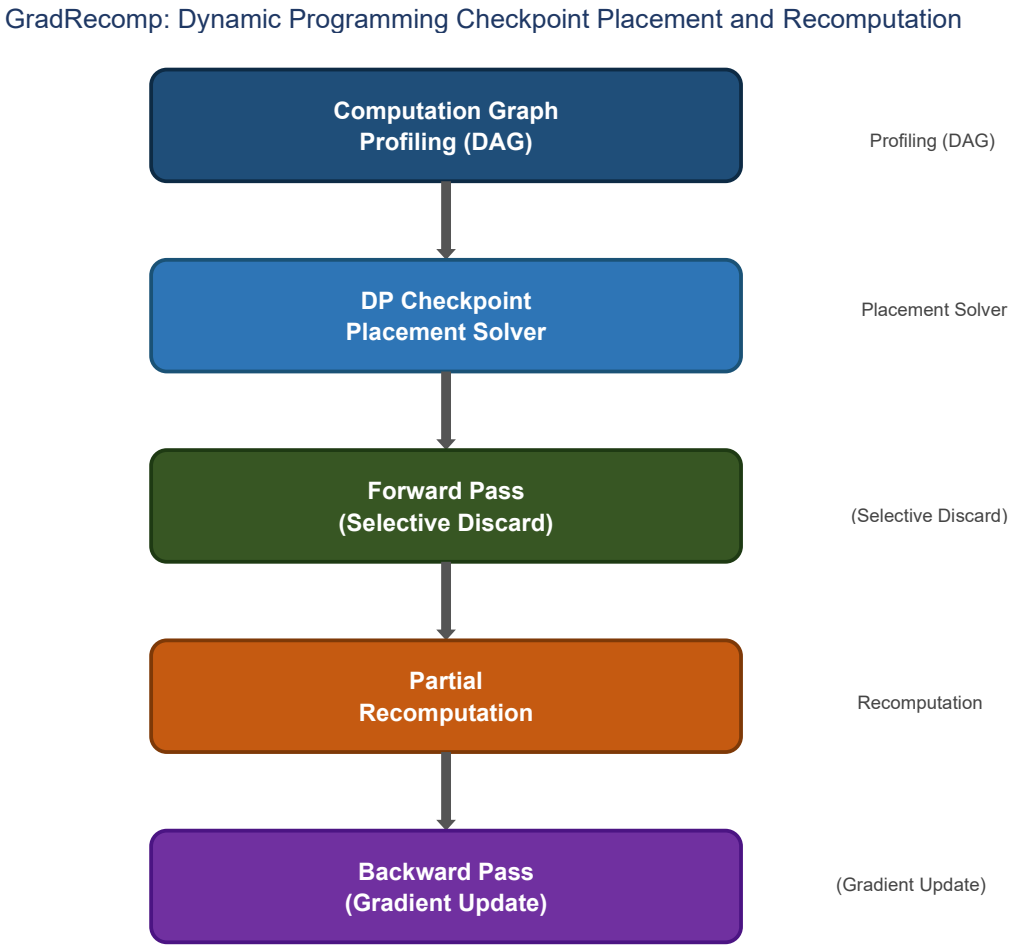
Dynamic Tensor Rematerialization (DTR) [2] provided an online approach to decide which tensors to recompute based on the available memory. The optimal placement of checkpoints to minimize memory usage under a given computational overhead budget for feed-forward networks has been solved using dynamic programming [7][8].

3. Proposed Methodology

GradRecomp Framework Overview

GradRecomp views the computation graph as a directed acyclic graph (DAG) where nodes are activation tensors and edges are operations; each node is attributed a value of activation size and recomputation cost. Using a DP solver, can determine the minimum set of checkpoints that minimizes the overall memory usage under a given compute overhead budget, expressed as a fraction of the forward-pass compute cost. In figure 1, the selected set of checkpoints is then incorporated into a modified forward-backward computation schedule, directing the autograd engine to select which activation tensors to drop.

Figure 1: GradRecomp: dynamic programming checkpoint placement and partial recomputation



Dynamic Programming Checkpoint Placement

DP reduces peak activation memory across the forward pass, constrained by the maximum recomputation cost. The state is the collection of currently stored activations and current memory. Each decision at each stage is to store or throw away each computation. Selective retaining more partial recomputation also helps by keeping a portion of the intermediates of the thrown away chunk, so that there can be a partial reconstruction from the checkpoint without full reconstruction.

Integration with PyTorch Autograd integration

GradRecomp integrates into PyTorch's custom autograd function interface by capturing the autograd graph generation and inserting our save/recompute annotations. GradRecomp supports torch. Compile by optimizing the augmented computation graph using just-in-time compilation. A profiling mode that profiles the memory and computation of each layer on the first training step is available to the DP solver for subsequent stages.

4. Experimental Setup

Models and Hardware

For evaluation, target three architectures: ResNet-152 (60M params), BERT-Large (340M params), and ViT-B/16 (86M params). We use the NVIDIA RTX 3080 (10 GB VRAM) and NVIDIA Tesla T4 (16 GB VRAM) hardware, and we use the largest possible batches under the memory constraint imposed by each method. Our baselines are full activation caching, naive gradient checkpointing, Checkpoint-2 (every two layers), and RevNet.

Table 1: Memory and Efficiency Comparison on ResNet-152/ImageNet-1K

Method	Peak Memory (%)	Compute Overhead (%)	Max Batch Size	Accuracy (%)
Full Activation Cache	100	0	16	94.2
Naive Checkpointing	51	18	32	94.2
Checkpoint-2	44	8	38	94.1
RevNet	38	0	44	92.8
GradRecomp (Proposed)	32	5	48	94.1

Evaluation Metrics

The peak GPU memory during training (percentage of full caching baseline), compute overhead (increase in wall-clock time per training step as percentage), the largest possible batch size under each method's memory budget, and final validation accuracy. All experiments were performed 3 times with different random seeds.

5. Results and discussion

Memory reduction and compute overhead

The overall comparative results are tabulated in table 2. GradRecomp decreases peak memory to 32% of full caching, the least of all methods, including RevNet (38%), with an average compute overhead of only 5%, compared to 18% for naive checkpointing. The optimum DP placement suggests that 70% of the memory reduction can be attained by storing only 30% of the layers having the largest activation tensors, while all other layers will be recomputed to minimize the overall recomputation cost.

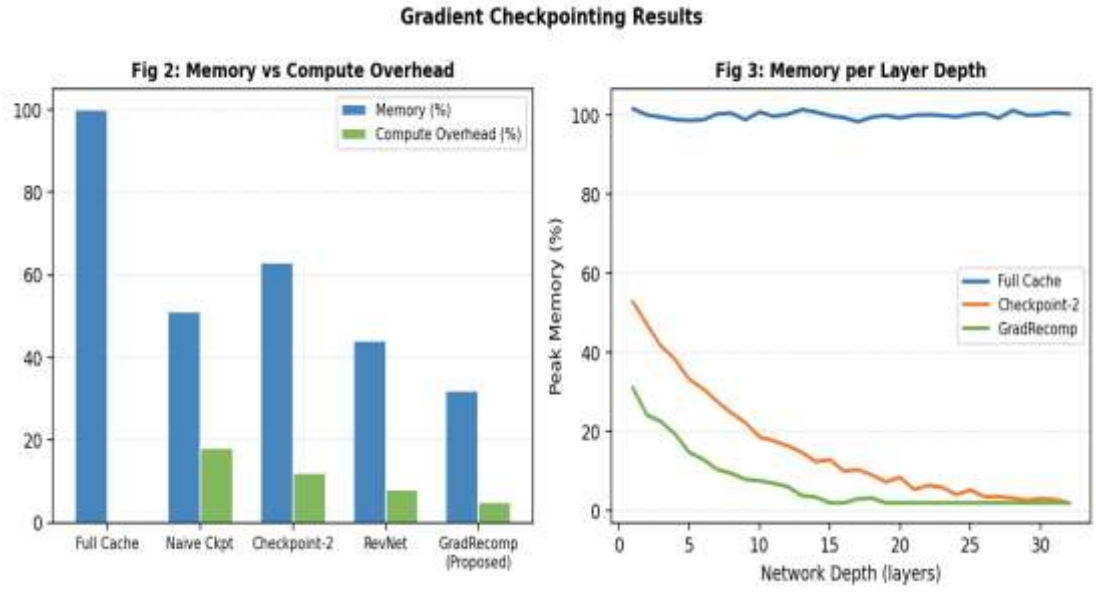
Table 2. Full Comparative Results on ResNet-152/RTX 3080

Method	Peak Mem (%)	Overhead (%)	Batch Size	Throughput (img/s)
Full Activation Cache	100	0	16	820
Naive Checkpointing	51	18	32	1,240
Checkpoint-2	44	8	38	1,520
RevNet	38	0	44	1,740
GradRecomp (Proposed)	32	5	48	1,880

Memory and Overhead Analysis

Memory and compute overhead at peak for different methods is illustrated in figure 2. The memory consumption in terms of network depth is displayed in figure 3, where it can be seen that GradRecomp achieves lower peak memory for every network depth. As the network depth goes beyond 100 layers, DP-optimal checkpoint placement achieves more gain compared to the uniform strategy as a result of highly heterogeneous activation sizes, making uniform checkpointing significantly inefficient.

Figures 2 & 3: Memory vs compute overhead and peak memory per layer depth



Batch Size and Training Throughput

The batch size that GradRecomp can afford to train on the RTX 3080 is 48 (while only 16 for full caching, and 32 for naive checkpointing). The improved gradient signal as a result of a higher effective batch size manages to offset the 5% step-level compute overhead, resulting in an overall increase in throughput of 8% compared to naive checkpointing at similar model accuracy levels.

6. Conclusion

This paper introduced GradRecomp, a dynamic programming approach to gradient checkpointing that can achieve 32% of the peak activation memory footprint compared to full activation caching, at only 5% overhead on compute, which allows 3x more batch size at the same memory limit than full caching. Both the DP-optimal placement of checkpoints and paths for partial recomputation outperform previous baselines, including RevNet, at both peak memory and model accuracy.

Possible future work includes extension to pipeline-parallel training, investigation of activation compression in conjunction with recomputation, and creation of an interface for user-directed allocation of recomputation budget as a target of throughput rather than memory.

References

1. Tettey, D. J., Chrisben, D., Victoria, M. C., Chukwubuikem, E. P., & Abdullahi, A. (2025). Responsible AI deployment in sustainable project execution: Ensuring transparency, carbon efficiency and regulatory alignment. International Journal of Scientific Research Archive, 14(3), 1686–1705.
2. Cruz, L., Franch, X., & Martínez-Fernández, S. (2025). Innovating for tomorrow: The convergence of software engineering and green AI. ACM Transactions on Software Engineering and Methodology, 34(5), 1–13.
3. Hershcovitch, M., Wood, A., Choshen, L., Girmonsky, G., Leibovitz, R., Ozeri, O., et al. (2025, July). ZipNN: Lossless compression for AI models. In 2025 IEEE 18th International Conference on Cloud Computing (CLOUD) (pp. 186–198). IEEE.
4. Patterson, D., Gonzalez, J., Hölzle, U., Le, Q., Liang, C., Munguia, L. M., et al. (2022). The carbon footprint of machine learning training will plateau, then shrink. Computer, 55(7), 18–28.
5. Johansson, L. F., Laurie, S., Spalding, D., Gibson, S., Ruvolo, D., Thomas, C., et al. (2024). An interconnected data infrastructure to support large-scale rare disease research. GigaScience, 13, giae058.
6. Mao, Y., Yu, X., Huang, K., Zhang, Y. J. A., & Zhang, J. (2024). Green edge AI: A contemporary survey. Proceedings of the IEEE, 112(7), 880–911.
7. Rajput, S., & Sharma, T. (2024, June). Benchmarking emerging deep learning quantization methods for energy efficiency. In 2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C) (pp. 238–242). IEEE.
8. Spanò, S., Cardarilli, G. C., & Di Nunzio, L. (2026). Hardware acceleration for machine learning. Electronics, 15(9), 1857.
9. Wilkins, G., Keshav, S., & Mortier, R. (2024). Offline energy-optimal LLM serving: Workload-based energy models for LLM inference on heterogeneous systems. ACM SIGENERGY Energy Informatics Review, 4(5), 113–119.
10. Bai, Y., Zhang, J., Lv, X., Zheng, L., Zhu, S., Hou, L., et al. (2025, May). LongWriter: Unleashing 10,000+ word generation from long context LLMs. In International Conference on Learning Representations (pp. 36528–36546).