WHAT IS THE STATE OF MEMORY SAVING FOR MODEL TRAINING?

Xiaoxuan Liu [1] **Chuyan Zhu** [1] **Jialun Lyu** [1] **Zhuohan Li** [1] **Xiaoyong Liu** [2] **Daniel Kang** [1] **Alvin Cheung** [1]

ABSTRACT

Large neural networks can improve the accuracy and generalization on tasks across many domains. However, this trend cannot continue indefinitely due to limited hardware memory. As a result, researchers have devised a number of memory saving methods (MOMs) to alleviate the memory bottleneck, such as gradient checkpointing, quantization, and swapping. In this work, we study memory saving methods and show that, although these strategies indeed lower peak memory usage, they can actually *decrease training throughput* by up to 9.3×. To provide practical guidelines for practitioners, we propose a simple but effective performance model PAPAYA to quantitatively explain the memory and training time trade-off. PAPAYA can be used to determine when to apply the various memory optimization methods in training different models. We outline the circumstances in which memory saving techniques are more advantageous based on derived implications from PAPAYA. We assess the accuracy of PAPAYA and the derived implications on a variety of machine models, showing that it achieves over 0.97 R score on predicting the peak memory/throughput, and accurately predicts the effectiveness of MOMs across five evaluated models on vision and NLP tasks.

1 INTRODUCTION

Deep neural networks (DNNs) are increasing in size and complexity as larger models improve the accuracy and generalization in machine learning applications. However, increasing model sizes is limited by the amount of device (typically a GPU) memory since the cost of storing feature maps and their gradients grows linearly with the *depth* (i.e., number of layers) and quadratically with the *width* (i.e., hidden state dimension) of the network.

In response, various memory optimization methods (MOMs) have been developed to reduce training memory footprint. Activation compressed training (ACT) compresses stored tensors to minimize saved activation during forward propagation, and most work uses quantization (Chakrabarti & Moseley, 2019; Fu et al., 2020; Chen et al., 2021; Liu et al., 2022) as the compression tool. Gradient checkpointing (Chen et al., 2016) trades computation for memory by discarding intermediate activations during forward propagation and recalculating them during the backward pass. Swapping (Peng et al., 2020; Huang et al., 2020; Wang et al., 2018; Kirisame et al., 2020) offloads GPU memory to the CPU when memory usage exceeds the device memory limit. Such methods introduce training overhead while reducing memory consumption for a given training batch size.

Preliminary work. Under review by the Machine Learning and Systems (MLSys) Conference. Do not distribute. [1]Department of Electrical Engineering and Computer Sciences, UC Berkeley [2]Alibaba Group US Inc. Correspondence to: Xiaoxuan Liu <xiaoxuan_liu@berkeley.edu>.

In this work, we survey the recent MOMs to understand when these techniques are helpful. We raise a number of questions: do those techniques really benefit training? Which technique achieves the best computation-memory trade-off? Are there strategies that work best on specific architectures or hardware?

We find that MOMs indeed help in two ways. First, they reduce the peak memory footprint compared to standard training for a fixed batch size. Second, they enable larger maximum batch sizes compared to standard training for a given hardware configuration.

Unfortunately, as we demonstrate, MOMs deliver mixed results when placed in the context of end-to-end training. Our experiments show that one major drawback of MOMs is that they can *slow down training by up to 9.3×*. Furthermore, one of the commonly used evaluation metrics, maximum batch size, does not directly demonstrate the effectiveness of MOMs, and can simply be increased via *gradient accumulation* (Ott et al., 2018) of smaller mini-batches, where we run forward and backward passes of different mini-batches sequentially and accumulate the gradients of each mini-batch.

Given our results, we argue that the existing evaluation metrics (e.g., maximum batch size given the hardware, reduced peak memory with a given batch size, throughput overhead given the memory threshold and batch size, etc.) are insufficient in indicating the costs and benefits of memory optimization for training. Since the goal of training is to lower the loss as quickly as possible, we propose to use *maximum training throughput* to evaluate the effectiveness of

various MOMs, as it can directly indicate the effectiveness of a training process.

Based on the maximum throughput, we conduct a measurement study on training various machine learning models, including convolutional neural networks (CNN) and transformer based models. We study different MOMs and also various model sizes to empirically understand the relationship among maximum throughput, batch size and model size. Aligned with literature, the maximum batch size is increased by all MOMs across the five examined models. However, the majority of MOMs add too much overhead to effectively assist training. We further conduct a case study on Bert and show that MOMs only begin to give a higher maximum throughput and accelerate the training process when the model is larger than a certain threshold (more than 34 layers or the network width is greater than 1344).

Unfortunately, it takes significant computation resources to measure the maximum training throughput of training with MOMs. A number of iterations must be performed in order to accurately measure the throughput for a particular batch size. This procedure must be repeated for various batch sizes in order to determine the maximum throughput. Finally, the same measurement process is needed to determine the maximum throughput for each MOM. More importantly, we anticipate a performance model that goes beyond estimating performance and is able to quantify the trade-off between memory and compute for the evaluation of MOMs.

In this work, we propose a performance model – PAPAYA, that can quantify the efficiency of MOMs based on small amount of profiling results. We demonstrate the validity and effectiveness of PAPAYA on various model architectures: ResNet-50 (He et al., 2016), Wide-ResNet-50 (Zagoruyko & Komodakis, 2016), Bert (Devlin et al., 2018), Swin (Liu et al., 2021), and GPT3 (Brown et al., 2020), while running them on different hardware configurations, including different number of GPUs (1, 2, 4, 8 GPUs). Experiments show that PAPAYA achieves an R score higher than 0.97 in predicting the maximum throughput and peak memory. PAPAYA also accurately predicts whether a certain strategy can accelerate the training process. Finally, starting from the PAPAYA, we mathematically derive and verify two implications where applying MOMs will be beneficial: when the trained model is significantly large and when training is distributed. We believe these findings will provide insights for future research on developing efficient MOMs.

In summary, we make the following contributions:

• In Sections 3 and 4, we show that previous evaluation metrics for evaluating MOMs, such as maximum batch size given memory budget, size of reduced peak memory given batch size, etc., are insufficient, as the benefits of larger batch sizes a memory method enables can be outweighed by the computational overhead it introduces. Instead, we evaluate MOMs by their maximum throughput improvement (record/s).

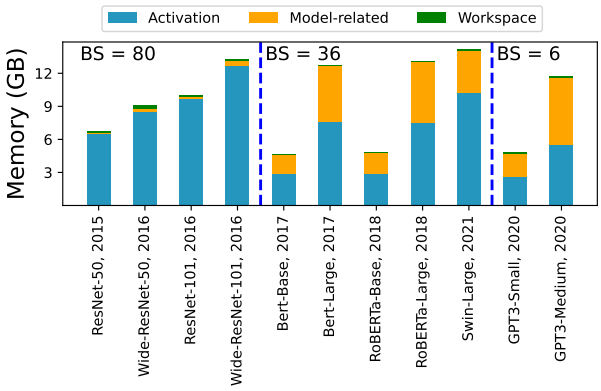

Figure 1. Memory consumption in different models. BS = Batch Size.

• In Section 5, we use maximum throughput as the metric and develop an intuitive performance model PAPAYA that allow users to quickly assess the effectiveness of each MOM. PAPAYA is simple and general enough to apply on a wide range of models, and quantifies the trade-off between memory and compute.

• In Section 6, derived from PAPAYA, we describe situations where applying MOMs are beneficial with theoretical proof and experimental verification.

2 OVERVIEW OF MEMORY OPTIMIZATION METHODS

In this work, we focus on *memory optimization methods*, which aim to reduce the total amount of memory used for DNN training on accelerators. The bulk of memory usage during training time falls under three categories: First, *activations* (i.e., feature maps) are the intermediate tensors stored during the forward pass that are needed to compute gradients during the backward pass. Second, *model related objects* that store the model's state, such as model parameters (e.g., weight matrices), optimizer states, and gradient accumulators, which collect gradients from earlier steps. For example, the momentum technique needs to calculate the parameter updates using both the gradients from the previous and the current step. Third, *workspace memory*, which includes temporary storage for computing calculations. For example, the CuDNN batch normalization backward function needs sufficient workspace memory to activate the fast NHWC semi-persistent kernel and speed up calculation (cud). Workspace memory will be released once the operator finishes execution.

We show the memory consumption of various components for a range of models in Figure 1. As demonstrated in the graph and mentioned in previous work (Jain et al., 2020),

activation takes up a significant portion of total memory for convolutional neural networks (CNNs). With the introduction of transformer-based model after 2017, the size of model-related objects becomes non-negligible.The majority of memory-saving solutions concentrate on lowering activation memory and they can be divided into three categories:

Gradient Checkpointing (Chen et al., 2016) trades computation for memory by dropping some of the activations during the forward pass from memory and recomputing them during the backward pass. Jain et al. (2020); Kumar et al. (2019) formalize the problem of trading computation for memory as a linear programming problem to generate a near-optimal schedule. To handle not only static but also dynamic computation graphs, Kirisame et al. (2020) proposes a greedy online algorithm for heuristically checkpointing arbitrary models. Korthikanti et al. (2022) combines gradient checkpointing with model parallelism for training transformer models.

Activation Compressed Training (ACT) compresses the training activation during the forward pass and decompresses the saved activation when calculating the gradient during back propagation. It has been applied to convolution neural networks using different compressors, such as quantizers (Chakrabarti & Moseley, 2019; Fu et al., 2020; Chen et al., 2021; Liu et al., 2022), JPEG (Evans et al., 2020), or scientific data compression algorithms (Jin et al., 2021; Evans & Aamodt, 2021). ACT has also been applied to transformers (Pan et al., 2021) and graph neural networks (Anonymous, 2022).

Swapping offloads activation or model parameters to external memory (e.g., CPU memory). (Wang et al., 2018; Huang et al., 2020; Peng et al., 2020) explore the search space of jointly optimizing operator scheduling, memory allocation, and swap decisions for static graphs. Recent work (Beaumont et al., 2021) also explores combining gradient checkpointing and swapping.

3 An Evaluation Metric for Memory Optimization Methods

We categorize existing MOM evaluation metrics in Section 3.1, and then describe their drawbacks. For example, max batch size, which is commonly used in prior work, simply measures reduction in memory consumption and ignores the computational overhead introduced by the given MOM. Therefore, it is difficult for users to determine if such optimizations should be used as both training speed and the peak memory consumption are crucial considerations when selecting MOMs. Perhaps surprisingly, as we will describe in Section 4, applying such optimizations can actually reduce maximum training throughput and therefore increase overall training time. Based on our observations, we propose a new metric – maximum throughput, which reflects the result of computation-memory trade off and its impact on overall training time, to evaluate the efficiency of MOMs in a straightforward manner.

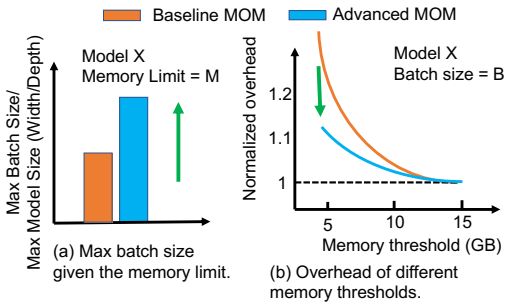

Figure 2. Existing evaluation metrics. The direction that a MOM is better under that metric is indicated by a green arrow.

3.1 Existing Evaluation Metrics

Previous evaluation of various MOMs can be categorized as follows: (1) Chen et al. (2021); Peng et al. (2020); Liu et al. (2022) compare the largest batch size the MOM enables given the same memory budget as shown in Figure 2(a), which typically is the memory capacity of the hardware. (2) Similarly, Chen et al. (2021); Liu et al. (2022) compare the largest model the MOM allows given the same batch size and memory budget, where the largest depth and width of the model are used to assess the efficacy of a particular MOM. (3) Kirisame et al. (2020); Chen et al. (2021); Huang et al. (2020); Jain et al. (2020); Liu et al. (2022) compare the performance overhead of a MOM with the original training under different memory thresholds, given the same batch size on a specified model, as shown in Figure 2 (b). Most prior work uses metrics (1) and (3).

3.2 Drawbacks of Existing Evaluation Metrics

As shown in Section 3.1, the evaluation metrics lack a direct and unified reflection of their effects on model training. First, all MOMs trade extra computation, more memory access or communication for reduced memory consumption. We argue that only taking memory reduction (such as the maximum batch size or maximum model size) into account is unfair as the overhead (i.e., extra training time) is ignored. As shown in Section 4, MOMs increase max batch size when evaluated using maximum batch size (e.g., some can increase batch size by up to 11.2×), but training is actually slowed down (50.6% max throughput of the original training, which means the overall training time is 1.97× of the original if the total number of training epochs keeps unchanged).

Moreover, the overhead given a batch size is also insuf-

Table 1. Notation used in this paper

Symbol	Description	Symbol	Description
m	Peak memory of training	α	Linear coefficient of m
t	Training latency of one batch	β	Constant coefficient of m
v	Throughput	γ	Linear coefficient of t
x	Batch size	δ	Constant coefficient of t
M	Device memory limit	T	Total training latency

ficient as it ignores the fact that MOM permits a larger batch size, which may speed up training due to increased hardware usage and amortized framework overhead, thus speeding up end-to-end training even though it introduces some computational or communication overhead.

3.3 A New Evaluation Metric

Considering the drawbacks of previously proposed metrics, we propose to use the maximum training throughput (i.e., number of records the training technique can process per second) given a fixed machine configuration as the quantitative indicator of a MOM's effectiveness. Maximum throughput is general enough to be adopted on any MOM and models. Given fixed epoch and data set size, if throughput v increases, then the total training latency T will decrease. An ideal MOM should boost model training by increasing higher throughput to reduce overall training time. For the following we will use the notations described in Table 1.

All existing MOMs trade extra computation (e.g., recompute evicted activations) or communication (e.g., swap between CPU and GPU) for memory reduction. Therefore, given the same batch size, all such approaches have a lower throughput than the original. However, with extra memory, training can be done using larger batch sizes. As illustrated in our case study, a larger batch size usually results in a higher throughput. As a result, such MOMs are useful only if the maximum batch sizes they can achieve contribute to a higher maximum throughput than training without MOM. Therefore, we propose to use maximum throughput as our evaluation metric in this work.

In this paper, we focus on the training memory and throughput trade off. We leave the accuracy trade off to future research and assume that training accuracy remains constant regardless of batch size. As assumed in prior work (Goyal et al., 2017; Kaplan et al., 2020; Li et al., 2021), training should converge if the learning rate grows proportionally to the batch size until the batch size becomes sufficiently large (8k in ImageNet), which is often unattainable given the amount of memory available in typical systems.

4 Evaluating MOMs

In this section, we compare the throughput of several MOMs on a variety of machine learning models. We begin by outlining the experimental setup, then introduce the tested models and assessed MOMs.

Model	Quantization	Gradient Checkpointing	Swapping	Combinations
ResNet-50	ActNN (Chen 2021)	DTR (Kirisame 2021)	Synchronous Swapping	Quantization + Swapping
Wide-ResNet-50				
Bert-Large		Checkpoint First (HuggingFace 2019)		Gradient Checkpointing + Swapping, Quantization + Swapping, Gradient Checkpointing + Quantization
Swin-Large		Checkpoint First (Hwang 2022)		Quantization + Swapping
GPT-Small		Checkpoint First (Fairseq 2019)		Quantization + Swapping Gradient Checkpointing + Swapping

Figure 3. MOMs evaluated in this paper

The majority of MOMs do allow for greater training batch sizes, as illustrated in Figure 4, but do so at the expense of training speed. In addition, we find that as the model size increases, the maximum throughput gap between MOMs and the original decreases. Based on the observation, we hypothesize that MOMs are better for larger models.

To verify the hypothesis, we apply MOMs to Bert with various widths and depths. We show that when the model is wide or deep enough, training with MOMs can indeed increase throughput, showing that only in specific circumstances are MOMs useful and that applying them haphazardly may actually harm training by increasing overall training time.

We conduct our study on the AWS p3.2xlarge instance (AWS, 2022) that has a single NVIDIA Tesla V100 GPU with 16 GB memory and 8 Intel Xeon Scalable (Skylake) vCPUs with 60 GB RAM.

4.1 Results on Various Models

Evaluated Models. We evaluate different MOMs on both convolutional neural networks (CNN) and transformer based models. For CNNs, we evaluate on ResNet-50 (He et al., 2016) and Wide-ResNet-50 (Zagoruyko & Komodakis, 2016). For transformer based models, we test both language tasks (Bert-Large (Devlin et al., 2018), GPT-Small (Radford et al., 2021)) and vision tasks (Swin-Large (Liu et al., 2021)).

Evaluated MOMs. We list all evaluated MOMs and their combinations in Figure 3. For the state-of-art activation compressed training, we utilize the ActNN (Chen et al., 2021) quantizer, which quantizes each element of the activation layer into four bits. For gradient checkpointing on CNNs, we choose DTR (Kirisame et al., 2020), a greedy online algorithm that evicts activations based on different heuristics when the peak memory exceeds the hardware limit. We test mainstream checkpointing policy that checkpoints the input of each transformer block for transformer based models. We select the checkpoint first policy for transformer based models because it has the best support

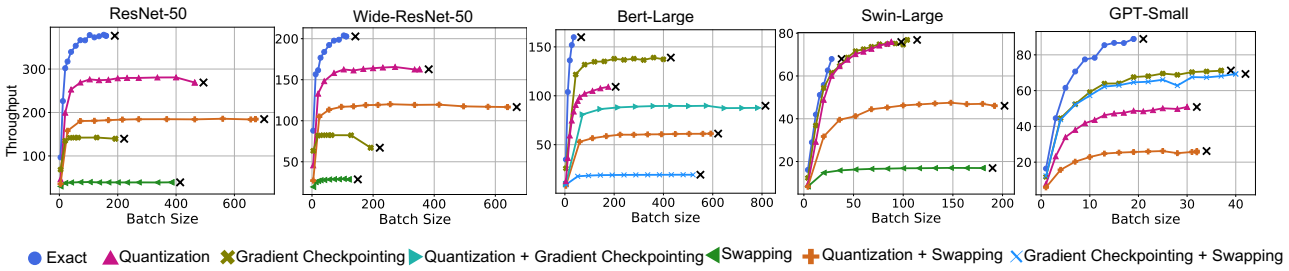

Figure 4. Training with different MOMs. The cross means a bigger batch size will cause the out of memory error.

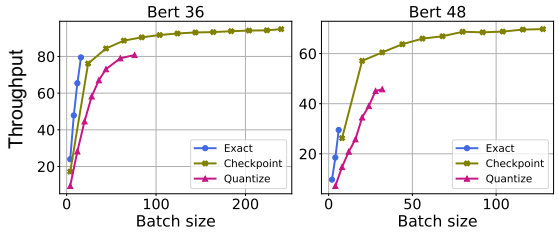

Figure 5. Bert training throughput with 36 layers and 48 layers.

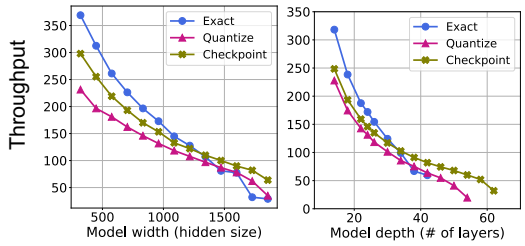

Figure 6. Bert training throughput with various widths and depths.

across different models and frameworks, thus adopted most widely. For swapping, although we are aware of different swapping techniques as described in Section 2, it is difficult to find an open-source implementation that is not based on simulation and provides good support for tested networks and frameworks. Therefore, we implement synchronous swapping in PyTorch ourselves. Additionally, we test the combinations of different MOMs as shown in the Figure 3.

Findings. As shown in Figure 4, the throughput of most training instances increases with batch size, although to varying degrees. The only exception is DTR on Wide-ResNet-50, where the maximum throughput drops from 80.6 records/s to 65.1 records/s as batch size increases from 160 to 200. This is because DTR determines the rematerialization policy based on the memory limit dynamically, and evicts more activations as batch size increases and memory budget decreases.

All training instances display diminishing return as batch size increases. For the original training on Bert-Large and Swin-large, the training process is aborted due to out of memory error before reaching a throughput plateau, whereas all other training instances reach a plateau where increasing the batch size barely increases training throughout.

Lastly, when compared to the original training, all MOMs increase the maximum batch size. On average, different MOMs can increase the maximum batch size by 2.6×, 2.8×, 11.2×, 4.5×, 1.8× for the five evaluated models, which aligns with previous work.

However, when considering the purpose of efficient model training as mentioned above, the maximum throughput should be the primary criterion. And only gradient checkpointing and quantization on Swin-Large increase the maximum throughput by 8.8% (76.86 vs 70.65) and 6.7% (75.35 vs 70.65) respectively. On average, the maximum training throughput are only 43.6%, 46.1%, 50.6%, 76.6%, 64.4% of the original on five evaluated models when MOMs are applied. In this case, the two metrics (maximum batch size vs. maximum throughput) lead to completely opposite conclusions: MOMs are promising when evaluated using maximum batch size as the metric, while training is actually slowed down when evaluated using maximum throughput, which argues that the majority of MOMs are actually detrimental to training.

4.2 Case Study on Bert

Next, we ask if MOMs can be used to improve *both* batch size and throughput. From our results shown in Figure 4, training with quantization and gradient checkpointing are the most promising as they achieve the highest maximum throughput among all MOMs. Therefore, we apply those two MOMs on various Bert model sizes to see if they improve the maximum throughput. We first conduct experiments on Bert models with 36 layers and 48 layers. As shown in Figure 5, for Bert with 36 layers, the maximum training throughput with gradient checkpointing marginally exceeds that of the original. For the Bert 48 model, both training with quantization and checkpointing improve throughput, improving maximum throughput by 80.1% (38.7 vs 69.7) and 32.3% (38.7 vs 51.2) respectively.

Based on our findings, we hypothesize that MOMs are more

advantageous for larger models. To verify our hypothesis, we experiment with Bert models of various widths (hidden size ranges from 384 to 1920) and depths (layer number ranges from 8 to 62) and investigate the max throughput while training using MOMs. As shown in Figure 6, for both the original training and training with different MOMs, the maximum throughput decreases as model size increases (either depth or width). Additionally, the original training outperforms the memory optimized ones when model size is relatively small, and there is a critical point in model size where a certain MOM starts to increase throughput. Concretely, only when the Bert model is deeper than 34 layers or is wider than 1344 with respect to the hidden size do MOMs start to display benefits for training speed.

From these results, we find that MOMs are only beneficial under limited scenarios and can even degrade training throughput. Therefore, we want an efficient, accurate and intuitive cost model to evaluate the benefits of MOMs in improving training throughput. The cost model should be simple and general enough such that it can be easily used before training starts to determine if a MOM would be beneficial. Such model can bring tremendous benefits to reduce unnecessary waste of valuable computational resources in training deep learning models.

5 COST ESTIMATION

To aid practitioners in deciding which MOMs to apply for their workloads, we develop a new performance model called PAPAYA. PAPAYA predicts the performance of MOMs by estimating their effect on memory consumption and training throughput. In this section, we describe PAPAYA by first discussing our cost models for the two components, followed by the evaluation of PAPAYA's prediction accuracy.

5.1 Performance Model

As mentioned in Section 3, we use maximum throughput as the evaluation metric for MOMs. For MOM to speed up training, the maximum throughput of the MOM (v_{mom}) should be higher than that of the original training (v_0), which can be expressed as:

$$\max(v_{mom}) \geq \max(v_0) \tag{1}$$

Next, as training throughput is defined as the number of the records processed per second, it can be expressed as batch size divided by batch latency:

$$v = \frac{x}{t(x)} \tag{2}$$

Meanwhile, the peak memory consumption of both the original training and MOM cannot exceed the device limit M when achieving the maximum throughput:

$$\arg\max_x(v_0) = \arg\max_x \left\{ \frac{x}{t_0(x)} \;\middle|\; m_0(x) \leq M \right\} \tag{3}$$
$$\arg\max_x(v_{mom}) = \arg\max_x \left\{ \frac{x}{t_{mom}(x)} \;\middle|\; m_{mom}(x) \leq M \right\}$$

To check if a MOM satisfies Inequality 1 under the memory constraint, we need a good estimator for memory $m(x)$ and batch latency $t(x)$.

5.2 Memory Estimation

Memory usage during training can be divided into two parts. The first part is the fixed memory consumption that is unrelated to batch size, which includes model parameters, optimizer states, and the framework overhead. The second part, such as activation and workspace memory, increases in proportion to batch size. As a result, a basic linear model, can be used to model the relationship between batch size and peak memory use:

$$m(x) = \alpha x + \beta \tag{4}$$

Here, the incremental memory cost of increasing batch size is represented by α, and the fixed memory consumption is represented by β. Depending on the amount of memory saved, different MOMs have different α's. If the incremental memory cost is α_0 for the original and α_{mom} for the memory saving approach, the incremental memory reduction is $\alpha_0 - \alpha_{mom}$ for the given MOM, and a smaller α_{mom} indicates higher memory reduction.

Memory model validation. Figure 7 shows an example of our memory model fit to measured peak memory values for a range of models and batch sizes. Each deep learning task is implemented using PyTorch (Paszke et al., 2019) and the peak memory is calculated with the `max_memory_allocated` API during each iteration. Overall, we find that our model matches the observed data of both the original training and multiple MOMs closely while varying the batch sizes on different models, with an R score of over 0.97. With one exception, the peak memory of DTR (gradient checkpointing in ResNet-50 and Wide-ResNet-50) is not adequately modeled using our linear model. This is because DTR uses a dynamic checkpointing method that evicts different activations under different batch sizes. Since the batch size is the only parameter used in our linear model, it is assumed that the rematerialization approach is the same for all batch sizes. Therefore, the linear model cannot accurately fit the peak memory of DTR.

Lastly, due to memory fragmentation, there is a discrepancy between peak memory and device memory (16 GB) for both the original and MOMs. A large batch size means that a large chunk of contiguous memory is requested all at once,

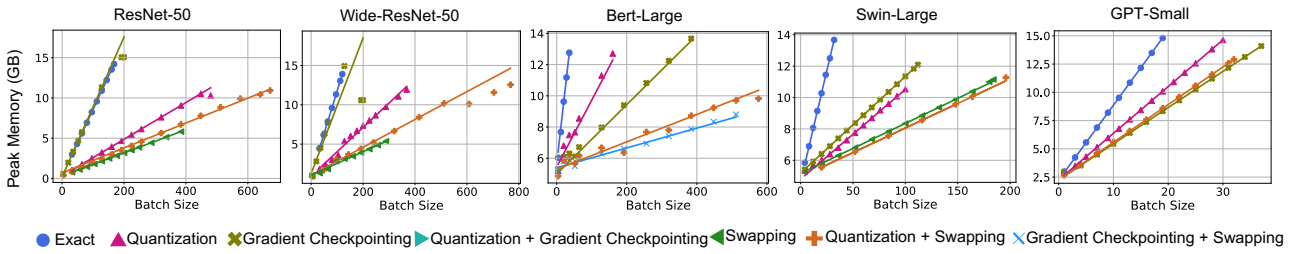

Figure 7. The points represents the profiled peak memory use. The solid lines are the PAPAYA predicted results with 20% of profiled data.

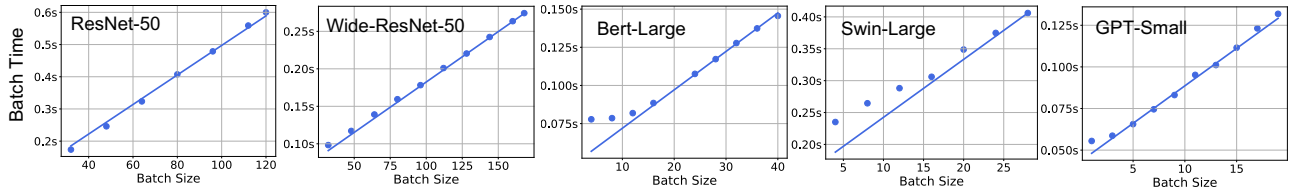

Figure 8. The points represents the profiled batch latency. The solid lines are the PAPAYA predicted results with 20% of profiled data.

which increases the likelihood that memory fragmentation ultimately causes the out of memory error. Main memory capacity is another factor that contributes to the discrepancy between the peak and device memory for swapping. For instance, as the offloaded memory is too large to fit in main memory, applying swapping on ResNet-50 and Wide-ResNet-50 uses nearly all of main memory despite peak GPU memory usage is still very low (5.8G for ResNet-50 and 5.3G for Wide-ResNet-50).

5.3 Throughput Estimation

Similarly, we use a linear model for the relationship between batch size and training latency to train one batch:

$$t(x) = \gamma x + \delta \tag{5}$$

Meanwhile, the incremental execution cost of increasing batch size is γ, while the fixed execution cost is δ, which includes time spent on updating the weights, framework overhead, etc. If the incremental cost is γ_0 for the original and γ_{ms} for the memory saving approach, the incremental overhead is $\gamma_{ms} - \gamma_0$ for the given memory saving strategy, and a smaller γ_{ms} indicates lower overhead.

We recognize that this may not be perfect as the execution latency is not a linear function of batch size when the GPU is underutilized. When the GPU is underutilized, increasing the batch size will increase GPU utilization instead of introducing more computation time, resulting in a sub-linear batch latency. However, as shown in Figure 8, such non-linearity only happens when the batch size is very small, which is not the regime of most of the experiments in the paper.

Batch latency model validation. Figure 8 shows the batch latency across different batch sizes for the original training on a variety of training tasks. For transformer-based models, the batch latency displays non-linearity when the batch size is small. And after the batch size reaches a certain value (e.g., 16 for Bert-Large), the linear model fits the batch latency closely. Furthermore, for all evaluated models, the batch latency establishes linearity at a small batch size. Thus, we only need to avoid a small under-utilized range of batch sizes when sampling to make the prediction of our performance model accurate and robust.

Next, we substitute $t(x)$ into Equation 2 and get a simplified expression for throughput:

$$v = \frac{x}{\gamma x + \delta} = \frac{1}{\gamma + \delta/x} \tag{6}$$

As shown in Equation 6, throughput is a monotonic function of batch size and a larger batch size leads to higher throughput. Moreover, the increased throughput comes from the reduction of amortized fixed execution cost ($\frac{\delta}{x}$) as we increase the batch size. If we take the derivative of Equation 6, we get:

$$v' = \frac{\delta}{\gamma x^2 + \delta} \tag{7}$$

Equation 7 shows a diminishing return as batch size is increased, where the throughput growth rate decreases quadratically with the batch size. This also partially explains why all those MOMs discussed earlier fail to outperform the original throughput: the benefits of memory reduction (which is demonstrated by the increased batch size) wear off too quickly before they provide any throughput increase.

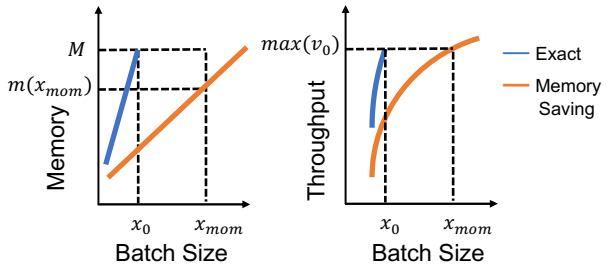

Figure 9. Derivation of the relationship between batch sizes and memory consumption

5.4 Performance Model Simplification and Verification

Once we accurately estimate the memory and throughput, we can leverage them to solve Inequality 1 to determine if a given OOM should be applied.

First, as indicated in Equation 6, throughput is a monotonic function of batch size. We need to increase the batch size as much as possible to obtain the best throughput. Device memory, on the other hand, limits the largest batch size possible. As shown in Figure 9, given device memory limit M, with Equation 8, we can calculate the value of x_0 such that the original training achieves the maximum throughput $\max(v_0)$, as shown in Equation 9:

$$\alpha_0 x_0 + \beta_0 = M \tag{8}$$

$$\max(v_0) = \frac{x_0}{\gamma_0 x_0 + \delta_0} \tag{9}$$

Next, as illustrated in Figure 9, we can find a x_{mom} such that the MOM achieves the same throughput as the $\max(v_0)$ as represented in Equation 10.

$$v_{mom} = \frac{x_{mom}}{\gamma_{mom} x_{mom} + \delta_{mom}} = \max(v_0) \tag{10}$$

Compared with the original training, to increase batch size and get an equal or greater throughput for MOM, the memory cost m_{mom} at batch size x_{mom} should not exceed the device memory capacity, i.e., $m_{mom}(x_{mom}) \leq M$.

After substituting Equation 9 and Equation 10 into Inequality 3, we get:

$$\frac{\delta_{mom}\alpha_{mom}}{(\gamma_0 - \gamma_{mom})(M - \beta_0) + \delta_0\alpha_0} \leq \frac{M - \beta_{mom}}{M - \beta_0} \tag{11}$$

Furthermore, as the fixed training latency overhead should be the same for the original and different MOMs, we can assume that

$$\beta_0 = \beta_{mom} = \beta \tag{12}$$

Similarly, we further assume that the fixed execution cost is the same in both cases, in other words:

$$\delta_0 = \delta_{mom} = \delta \tag{13}$$

Finally, we can simplify Inequality 11 to:

$$\frac{\alpha_0 - \alpha_{mom}}{\gamma_{mom} - \gamma_0} \geq \frac{M - \beta}{\delta} \tag{14}$$

Inequality 14 is interesting in that it is independent of batch size. It is determined only by γ_0, γ_{mom}, α_0, α_{mom}, δ, β, M, which are all fixed once the MOM, hardware, and model architecture are chosen.

Moreover, Inequality 14 is very intuitive. Since α is the incremental memory cost of increasing the batch size, $\alpha_0 - \alpha_{mom}$ is just the incremental memory saving provided by the MOM. Similarly, γ represents the incremental execution cost of increasing the batch size, hence $\gamma_{mom} - \gamma_0$ is the overhead incurred by the MOM. Inequality 14 implies that *training with a given MOM is beneficial only when the ratio between memory savings and overhead imposed by the MOM is larger than a threshold established by the fixed memory and execution cost.*

We define the left side of Inequality 14 as a score to evaluate the efficiency of a memory optimization method, and we call that the PAPAYA Score:

$$P = \frac{\alpha_0 - \alpha_{mom}}{\gamma_{mom} - \gamma_0} \tag{15}$$

In the score, the denominator indicates how much memory a given MOM can save, while the computational overhead is the numerator. The higher the PAPAYA Score, the more memory can be saved for the same amount of overhead, and the memory saved also translates to significant throughput improvement. A lower PAPAYA Score, on the other hand, indicates that more work is needed to save the same amount of memory. Even if the potential batch size is larger, the overhead wears off the advantages when the PAPAYA Score falls below a threshold and the MOM fails to accelerate training. We define this threshold as the PAPAYA Point $\hat{P}$, shown in Equation 16:

$$\hat{P} = \frac{M - \beta}{\delta} \tag{16}$$

$\hat{P}$ is only related to M, β and δ, which are determined by device and model but not affected by the batch size.

Performance model validation Next, we evaluate if the PAPAYA Score is a good indicator of training speed. As can be seen from Figure 10, for all cases, when the normalized P score is greater than one (i.e., the PAPAYA Score is higher than the PAPAYA Point), the MOM also has a higher maximum throughput than the original training (normalized maximum throughput is greater than one). Additionally, the PAPAYA Score is a monotonic function of the maximum throughput. Therefore, users can use the PAPAYA Score to compare different MOMs as a higher PAPAYA Score also indicates a higher max throughput.

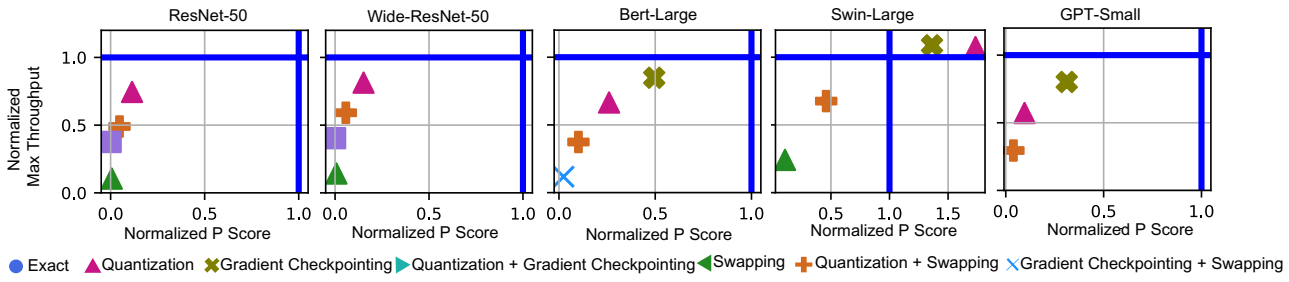

Figure 10. Normalized PAPAYA Score and normalized maximum throughput on five evaluated models. Normalized maximum throughput = maximum throughput of MOM / maximum throughput of the original training. Normalized P score = PAPAYA Score of MOM / PAPAYA Point.

5.5 Limits of the performance model

First, as mentioned in Section 5.2, the derivation of the PAPAYA Score and PAPAYA Point does not take memory fragmentation into account. However, we can modify Inequality 14 to make them adapt to different fragmentation levels as we describe in Appendix A. The modified formula takes the memory utilization ratio as the input. We also describe how binary search can be used to find the maximum efficient memory. However, as shown in Section 5.4, the current PAPAYA Score/Point is accurate enough to make a good prediction of different MOMs' maximum throughput without taking memory fragmentation into account.

Second, our model uses a linear function to predict latency, which is insufficient when the batch size is small (i.e., when the GPU is underutilized). However, as shown in Section 5.3, current neural models typically fully utilize the hardware even with a small batch size, and hence our linear function can already model the latency in most cases. Moreover, the current model assumes that the batch size is the only factor that affects the peak memory and batch latency, therefore it does not handle dynamic optimization strategies such as DTR as mentioned in Section 5.2.

6 IMPLICATIONS

In this section, we show that the PAPAYA Score and PAPAYA Point can be used to predict scenarios where MOMs can improve max throughput. We state our implications followed by a theoretical proof derived from Formula 14, then we show experiment results to further validate our implication. For all experiments in this Section, we pick the best performing MOM in Figure 4 for a given model.

6.1 Preference for large models

MOMs are more beneficial for larger models. Here, we focus on models within the same family (e.g., Bert models of different layers, GPTs with different numbers of encoder/decoder blocks, etc) as models from different families are not directly comparable.

Proof: (1) Model depth: If the depth of a model increases (e.g., add the more transformer blocks), both the incremental memory cost and incremental batch latency increase linearly with the model depth (i.e., $\alpha = C1 \cdot D, \gamma = C2 \cdot D$, where $C1, C2$ are constants and D is the depth of the network). If we increase the model depth by $k\times$, the incremental cost scales $k\times$ as well for both the original training and different MOMs. Putting the new αs and γs into Inequality 14, the left part keeps unchanged since the k cancels out for the numerator and denominator. On the other hand, the fixed memory cost increases (β) as the model gets larger. Therefore, the PAPAYA point is lower and the PAPAYA score keeps unchanged. As a result, while increasing the model depth, it is easier for PAPAYA score to surpass PAPAYA point, which means MOMs are more likely to improve the maximum throughput.

(2) Model width: If the width (e.g., the hidden size of each layer) of a model increases, the incremental memory cost will increase linearly with the width, while the incremental batch time will increase quadratically with the model width (i.e. $\alpha = C1 \cdot W$, $\gamma = C2 \cdot W^2$, where $C1$ and $C2$ are constants and W is the width of the network). After putting the incremental memory/batch time cost into Equation 15, we get the new PAPAYA Score:

$$\frac{C1_0 - C1_{mom}}{(C2_{mom} - C2_0) \cdot W} \tag{17}$$

Meanwhile, the fixed memory cost (β in Formula 16) increases quadratically with the model (i.e., $\beta = C'1 \cdot H^2$, where $C1'$ is a constant and W is the width of the network) because the model-related objects (e.g., model parameters, optimizer states) occupy most of the fixed memory consumption and they increase quadratically with the model width. Similarly, as the fixed time cost is mainly caused by the parameters updates, the fixed batch time also increases quadratically with the model width (i.e., $\delta = C'2 \cdot H^2$,

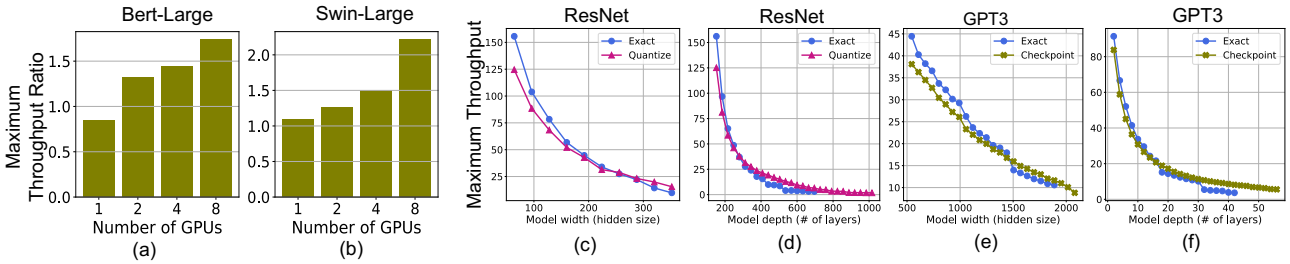

Figure 11. (a), (b) show the maximum throughput ratio of training with 1, 2, 4, 8 V100 GPUs on Bert-Large and Swin-Large. (c) - (f) shows the maximum throughput of training ResNet and GPT with different depths and widths.

where $C'2$ is a constant). After putting them into the PAPAYA Point formula (Formula 16), we get the new PAPAYA Point:

$$\frac{M - C1' \cdot W^2}{C2' \cdot W^2} \tag{18}$$

Comparing the new PAPAYA Score and PAPAYA Point, we get the inequality

$$\frac{C1_0 - C1_{mom}}{(C2_{mom} - C2_0)} > \frac{M - C1' \cdot W^2}{C2' \cdot W} \tag{19}$$

From the inequality, we find that the left part is unrelated with the model width and right part becomes smaller as the model width increases, indicating that it is easier for the PAPAYA Score to exceed the PAPAYA Point.

Verification: We verify this implication by varying the model depth and width for ResNet and GPT3 models on V100. As shown in Figure 11, when the model is shallow (less than 280 layers in ResNet, less than 18 layers in GPT3) or thin (hidden size is smaller than 248 for ResNet, 1500 for GPT3), applying MOMs actually slows down training. However, as the model gets deeper or wider, MOMs start to speed up training. In the extreme case, the original training fails to run even with batch size of one and applying MOM is needed to enable single GPU training.

6.2 Preference for multiple GPUs

MOMs are more beneficial for multi-GPU data parallel training. The intuition behind the implication is that communication makes MOMs less expensive.

Proof: Communication and synchronization across multiple devices result in a substantially higher fixed execution cost for multi-GPU training. According to PAPAYA Point in Equation 16, δ (the portion of batch time independent of batch sizes) increases because communication costs in data parallel setting is included in δ, and the synchronous overhead grows when more GPUs are used. As a result, the PAPAYA Point is lower and and it is easier for the PAPAYA Score to surpass the PAPAYA Point.

Verification: We compare the maximum throughput ratio between gradient checkpointing and the original training on Bert-Large and Swin-Large on distributed data parallel training with 1, 2, 4, 8 V100 GPUs. As shown in Figure 11, the ratio increases by adding more GPUs. Specifically, on Bert-Large, gradient checkpointing slows down training on a single GPU setting. However, as we increase the number of GPUs, gradient checkpointing achieves a higher maximum throughput than the original training, and can speedup the training process by $1.7\times$ when there are 8 GPUs.

7 CONCLUSION

In this paper, we argue that existing evaluation metrics for MOMs are insufficient and propose to use the maximum throughput to evaluate different MOMs. Based on the new metric, we investigate the effectiveness of different MOMs on various models and perform detailed analysis on Bert. We find that existing MOMs bring limited benefits for training deep learning models, where larger batch sizes does not necessarily translate to faster training time. We then propose linear models to predict the peak memory use and batch execution latency. The linear model partially explains the inefficiency of those MOMs: the throughput increases as we have a bigger batch size, but the benefits of memory reduction (as demonstrated by bigger batch size) wear off too quickly to result in any throughput increase. We capture this insight in a new metric called the PAPAYA Score, and use it to evaluate different MOMs while training a number of deep learning models. The PAPAYA Score is a good indicator of their efficiency. Starting from PAPAYA, we also provide insights for future research on developing MOMs by identifying cases where applying MOMs can improve maximum throughput.

REFERENCES

Bacth normalization kernel description. https://docs.nvidia.com/deeplearning/cudnn/api/index.html#cudnnBatchNormalizationBackwardEx. Accessed: 2022-10-08.

Anonymous. EXACT: Scalable graph neural networks train-

ing via extreme activation compression. In *Submitted to The Tenth International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=vkaMaq95_rX. under review.

AWS. Amazon EC2 P3 Instances. https://aws.amazon.com/ec2/instance-types/p3/, 2022.

Beaumont, O., Eyraud-Dubois, L., and Shilova, A. Efficient combination of rematerialization and offloading for training dnns. *Advances in Neural Information Processing Systems*, 34:23844–23857, 2021.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Chakrabarti, A. and Moseley, B. Backprop with approximate activations for memory-efficient network training. *Advances in Neural Information Processing Systems*, 32, 2019.

Chen, J., Zheng, L., Yao, Z., Wang, D., Stoica, I., Mahoney, M., and Gonzalez, J. Actnn: Reducing training memory footprint via 2-bit activation compressed training. In *International Conference on Machine Learning*, pp. 1803–1813. PMLR, 2021.

Chen, T., Xu, B., Zhang, C., and Guestrin, C. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Evans, R. D. and Aamodt, T. Ac-gc: Lossy activation compression with guaranteed convergence. *Advances in Neural Information Processing Systems*, 34, 2021.

Evans, R. D., Liu, L., and Aamodt, T. M. Jpeg-act: accelerating deep learning via transform-based lossy compression. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 860–873. IEEE, 2020.

Fu, F., Hu, Y., He, Y., Jiang, J., Shao, Y., Zhang, C., and Cui, B. Don't waste your bits! squeeze activations and gradients for deep neural networks via tinyscript. In *International Conference on Machine Learning*, pp. 3304–3314. PMLR, 2020.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Huang, C.-C., Jin, G., and Li, J. Swapadvisor: Pushing deep learning beyond the gpu memory limit via smart swapping. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 1341–1355, 2020.

Jain, P., Jain, A., Nrusimha, A., Gholami, A., Abbeel, P., Gonzalez, J., Keutzer, K., and Stoica, I. Checkmate: Breaking the memory wall with optimal tensor rematerialization. *Proceedings of Machine Learning and Systems*, 2:497–511, 2020.

Jin, S., Li, G., Song, S. L., and Tao, D. A novel memory-efficient deep learning training framework via error-bounded lossy compression. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 485–487, 2021.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Kirisame, M., Lyubomirsky, S., Haan, A., Brennan, J., He, M., Roesch, J., Chen, T., and Tatlock, Z. Dynamic tensor rematerialization. *arXiv preprint arXiv:2006.09616*, 2020.

Korthikanti, V., Casper, J., Lym, S., McAfee, L., Andersch, M., Shoeybi, M., and Catanzaro, B. Reducing activation recomputation in large transformer models, 2022. URL https://arxiv.org/abs/2205.05198.

Kumar, R., Purohit, M., Svitkina, Z., Vee, E., and Wang, J. Efficient rematerialization for deep networks. *Advances in Neural Information Processing Systems*, 32, 2019.

Li, Z., Zhuang, S., Guo, S., Zhuo, D., Zhang, H., Song, D., and Stoica, I. Terapipe: Token-level pipeline parallelism for training large-scale language models. In *International Conference on Machine Learning*, pp. 6543–6552. PMLR, 2021.

Liu, X., Zheng, L., Wang, D., Cen, Y., Chen, W., Han, X., Chen, J., Liu, Z., Tang, J., Gonzalez, J., Mahoney, M., and Cheung, A. GACT: Activation compressed training for generic network architectures. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 14139–14152. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/liu22v.html.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10012–10022, 2021.

Ott, M., Edunov, S., Grangier, D., and Auli, M. Scaling neural machine translation, 2018. URL https://arxiv.org/abs/1806.00187.

Pan, Z., Chen, P., He, H., Liu, J., Cai, J., and Zhuang, B. Mesa: A memory-saving training framework for transformers. *arXiv preprint arXiv:2111.11124*, 2021.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Peng, X., Shi, X., Dai, H., Jin, H., Ma, W., Xiong, Q., Yang, F., and Qian, X. Capuchin: Tensor-based gpu memory management for deep learning. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 891–905, 2020.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pp. 8748–8763. PMLR, 2021.

Wang, L., Ye, J., Zhao, Y., Wu, W., Li, A., Song, S. L., Xu, Z., and Kraska, T. Superneurons: Dynamic gpu memory management for training deep neural networks. In *Proceedings of the 23rd ACM SIGPLAN symposium on principles and practice of parallel programming*, pp. 41–53, 2018.

Zagoruyko, S. and Komodakis, N. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

A APPENDIX

Modified PAPAYA Score/Point that takes memory fragmentation into consideration,

$$\frac{\alpha_0 * M_0 - \alpha_{mom} * M_{mom}}{\gamma_{mom} - \gamma_0} \geq \frac{M_0 * M_{mom}}{\delta} \tag{20}$$

where

$$\begin{aligned} M_0 &= M * f_0 - \beta \\ M_{mom} &= M * f_{mom} - \beta \end{aligned} \tag{21}$$

f_0 is the effective memory ratio. For example, if the memory fragmentation takes 20% of the device memory, then the effective memory ratio is 80%. Since the original training and MOMs have different implementations, they will have different effective memory ratios (f_0 and f_{mom}).