An Analysis of Collocation on GPUs for Deep Learning Training

Ties Robroek, Ehsan Yousefzadeh-Asl-Miandoab, Pınar Tözün
IT University of Copenhagen
titr,ehyo,pito@itu.dk

ABSTRACT

Deep learning training is an expensive process that extensively uses GPUs. However, not all model training saturates modern powerful GPUs. To create guidelines for such cases, this paper examines the performance of the different collocation methods available on NVIDIA GPUs: *naïvely* submitting multiple processes on the same GPU using multiple streams, utilizing *Multi-Process Service (MPS)*, and enabling the *Multi-Instance GPU (MIG)*. Our results demonstrate that collocating multiple model training runs yields significant benefits, leading to up to three times training throughput despite increased epoch time. On the other hand, the aggregate memory footprint and compute needs of the models trained in parallel must fit the available memory and compute resources of the GPU. MIG can be beneficial thanks to its interference-free partitioning but can suffer from sub-optimal GPU utilization with dynamic or mixed workloads. In general, we recommend MPS as the best-performing and most flexible form of collocation for a single user submitting training jobs.

CCS CONCEPTS

• **Computing methodologies** → *Artificial intelligence*; *Machine learning*; • **Hardware**; • **Computer systems organization** → **Parallel architectures**;

KEYWORDS

resource-aware deep learning, collocation on GPUs, MIG

ACM Reference Format:
Ties Robroek, Ehsan Yousefzadeh-Asl-Miandoab, Pınar Tözün. 2024. An Analysis of Collocation on GPUs for Deep Learning Training. In *4th Workshop on Machine Learning and Systems (EuroMLSys '24), April 22, 2024, Athens, Greece.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3642970.3655827

EuroMLSys '24, April 22, 2024, Athens, Greece

ACM ISBN 979-8-4007-0541-0/24/04...$15.00
https://doi.org/10.1145/3642970.3655827

1 INTRODUCTION

Today's GPUs are significantly more powerful than those of a decade ago. Modern GPUs, together with larger datasets, facilitate the exponential growth of deep learning models. Many data scientists, however, do not require large models in practice. For example, a problem may not have a large enough dataset to warrant a large model, or the ideal batch size for training the model may not be large enough[1] to utilize all of the GPU resources [2, 11, 12, 28]. This poses an hardware under-utilization issue [11, 31] when training neural networks as the training process usually takes exclusive access to a GPU. This problem gets exacerbated with each new GPU generation offering more hardware resources.

Workload collocation is a method for increasing hardware utilization by running multiple applications at the same time over the same hardware resources. That way, the device and its resources are shared among the collocated applications. While workload collocation is heavily studied for CPUs [8, 10, 17], its opportunities and challenges have been largely unexplored for modern GPUs. In addition, unlike CPUs, GPUs lack sophisticated resource-sharing methods such as virtual memory and fine-grained sharing.

Today, there are several methods for workload collocation on a GPU. Firstly, multiple processes can be assigned to the same GPU simultaneously without any explicit process management. Alternatively, the collocation can be more precisely managed, for example via NVIDIA's *Multi-Process Service (MPS)*. Finally, the latest generations of NVIDIA GPUs can be partitioned into fully isolated GPU instances at the hardware level via *Multi-Instance GPU (MIG)*.

This paper analyzes different ways of collocating deep learning model training on NVIDIA GPUs. Specifically, we investigate the strengths and limitations of the new MIG technology in contrast to the older methods. We characterize the performance of the above-mentioned collocation methods on an A100 GPU. We diversify our workload by considering three datasets (ImageNet, ImageNet64x64, Cifar10) representing different sizes (large, medium, small). Furthermore, we

[1]Data scientists in our lab routinely use less than half of the requested GPU resources during their model parameter exploration.

acknowledge that the current deep learning landscape employs a wide variety of model architectures. We investigate two popular convolutional models (ResNet, EfficientNetv2) and one transformer model (CaiT). Additionally, we collocate a recommender model with a vision model to demonstrate the merits of workloads containing models that stress different parts of the hardware. Our results highlight that:

- When model training is unable to utilize the full GPU on its own, i.e., when running on our small- and medium-sized training cases or cases that stress different parts of the GPU, training multiple models in collocated fashion presents considerable benefits. On the other hand, for large model training, collocation provides either limited improvements to throughput as the GPU becomes oversaturated or cause model training to crash when the available GPU memory is not big enough to hold the combined memory footprint of the collocated models.
- On all the combinations we evaluated, MPS performs better than naïve and MIG collocation, allowing single-user workloads to get the most out of the hardware with minimal setup required.
- MIG offers strict separation of the GPU's memory and compute resources across the collocated workloads, eliminating interference. It also allows multi-user collocation, unlike MPS, and can achieve higher energy efficiency when the partitions are set well. On the other hand, MIG requires creating hardware partitions a priori. For the cases of well-defined workloads, one can create the ideal MIG partitions and leverage MIG-based collocation. However, for more dynamic workloads where the workload mix changes over time, MIG would require re-partitioning to perform well, whereas other collocation methods still provide benefits.

2 BACKGROUND

This section first provides background on different methods of collocation. Then, we survey related work on workload collocation for deep learning.

2.1 Collocation on GPUs

A ***CUDA stream*** [1] is a sequence of operations that execute on the GPU (i.e., kernels and data transfers) in the order they are issued. While operations within a stream are guaranteed to execute in the prescribed order, operations in different streams can run concurrently. This concurrency helps with overlapping the stall time due to the data transfers between the host CPU and GPU in one stream with work from another stream. We call this type of workload collocation the ***naïve*** method since it offers a limited way for sharing GPU resources. This is because the streams have to share the GPU compute resources in a time-based manner rather than having resources explicitly dedicated for each stream.

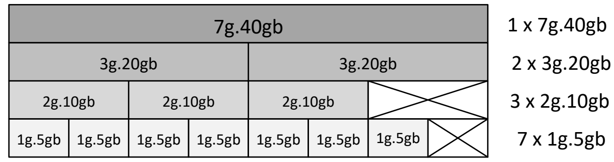


Figure 1: MIG partitioning schemes on a NVIDIA A100-40GB GPU. Horizontals can overlap but verticals cannot. For example, having a 3g.20gb instance is not compatible with five 1g.5gb instances but is compatible with two 2g.10gb instances (figure adapted from [18]).

The ***multi-process service (MPS)*** [20] enables the host CPU to launch multiple processes on a single GPU. Similar to naïve collocation, these processes share the GPU memory and memory bandwidth. However, unlike naïve collocation, the streaming multiprocessors (SMs) of the GPU are split across the different processes. Assignment of the SMs is done by the MPS daemon automatically, unless explicitly stated by the user, based on the provisioning of the GPU resources needed for each process. This reduces interference across the different processes compared to the naïve approach. One limitation of MPS is that it cannot collocate applications launched by different user accounts for security reasons.

Multi-instance GPU (MIG) [18] is the most recent collocation technology introduced with NVIDIA's Ampere GPUs. It provides hardware support for splitting a GPU into smaller GPU instances. Each instance can run a different process allowing these processes to run in parallel on the same GPU.

An A100 GPU with 40GB memory supports several available partitioning profiles (see Figure 1). The smallest possible GPU instance is one with just one memory slice and one compute slice, `1g.5gb`, with 14 streaming multiprocessors (SMs) and 5GB of memory. Consecutively, a `2g.10gb` profile consists of two compute slices (28 SMs) and two memory slices (10 GB of memory). The other available profiles are `3g.20gb`, `4g.20gb`, and `7g.40gb`. The last profile consists of almost all of the GPU resources. However, using the GPU without MIG mode is not analogous to running this large profile as the compute capability of the GPU is hampered slightly due to MIG management overhead; i.e. the reduced compute slice as mentioned above (10 SMs). Each partition is strictly separated in terms of hardware resources preventing any form of interference across partitions.

Many different partitions are possible as long as the maximum resource capacity is not exceeded. The partitioning rules are set by the GPU itself, and the allowed set of instances and configurations varies across different types of

Table 1: Models & Datasets

Model	Dataset	#Parameters	Size
ResNet26	Cifar10	17M	small
ResNet50	ImageNet64	24M	medium
ResNet152	ImageNet	59M	large
EfficientNet_v2_s	ImageNet64	22M	medium
CaiT_xxs24_224	ImageNet	12M	large
DLRM	Criteo Terabyte	24B	very large

NVIDIA GPUs (A100, A30, H100, H200). Finally, a GPU instance may also be split into multiple compute instances from the compute side with unified memory. This can be useful when compute and memory requirements do not follow the same pattern. For example, one could run a memory intensive model and a compute intensive model with isolated compute instances on a single GPU instance.

2.2 Related work

Collocation on GPUs have been studied in two dimensions: software and hardware approaches. Software approaches either focus on developing better primitives for collocation on GPUs or provisioning the resources of GPUs for running multiple applications [3, 22, 34]. In contrast, hardware approaches propose micro-architectural changes to GPUs to enable finer-grained and more precise multi-application execution within a GPU considering performance, utilization, and quality of service trade-offs [6, 7, 29, 30, 33, 36].

MIG is a relatively new technology and there have not been many works that thoroughly explore its possibilities. HFTA [28] is a mechanism to fuse multiple model training runs for hyper-parameter tuning into one training run. The authors show the effectiveness of HFTA compared to using MPS or MIG to run multiple training runs in parallel. MISO [15] runs MPS on a `7g.40gb` MIG instance to predict the best MIG configuration for different jobs. Finally, Li et al. [14] characterize performance of only MIG using deep learning models focusing on time and energy metrics.

In general, our work is orthogonal to these works since we investigate the strengths and limitations of MIG in contrast to the older collocation techniques such as MPS and naïve collocation and use workloads of different sizes.

3 IMPACT OF COLLOCATION

3.1 Setup & Methodology

System. Our experiments run on a DGX Station A100, composed of an AMD EPYC 7742 CPU (64 cores, 512GB RAM) and four A100 40GB GPUs (108 SMs). Each of the A100 GPUs have 40GB of VRAM and support up to 7 MIG instances with at least 5 GB of memory per instance (see Section 2.1).

Experiments. The experiments are devised with varying dataset sizes [4, 5, 13, 25] and models [9, 16, 26, 27] to assess the performance of collocating deep learning training under different loads (Table 1). We orchestrate the execution of the workloads via a benchmarking framework [23]. The vision models are sourced from the TIMM library [32], the recommender model from Facebook Research [16], and we are using the latest version of PyTorch as of the start of our experiments (2.0) [21].

3.2 Uniform Collocation

Figures 2-4 illustrate the results of our uniform collocation experiments. Each figure illustrates a particular model and dataset combination (as subset of the listed combinations in Table 1).[2] Bars that are grouped together form one collocated workload with models trained in parallel. The different degrees of collocation are separated by dotted vertical lines. The four non-collocated cases, which do not run any models in parallel, are the first four bars and form our baselines.

3.2.1 *Time per Epoch.*

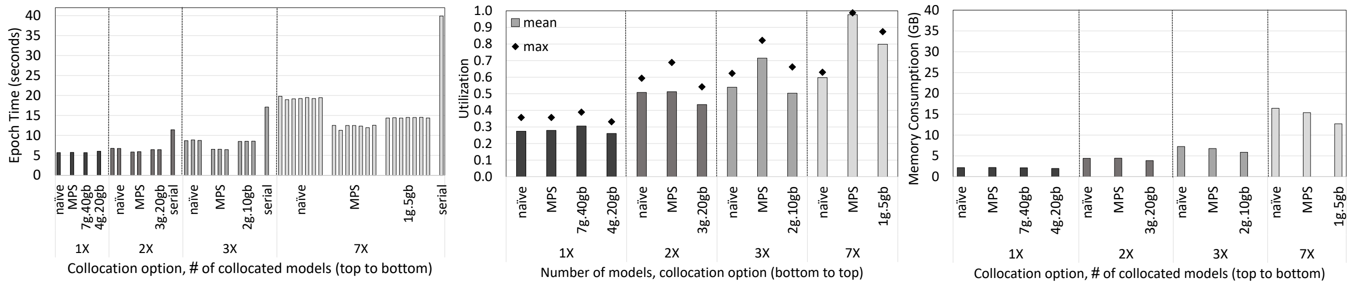
Our main performance metric when comparing the effectiveness of different collocation methods is ***Time per epoch***. We time the second epoch of training, skipping the first one as warm-up.

Looking at the first four bars of Figures 2a-4a, reveals that there is a little variation between the first three non-collocated workloads: `naïve`, `mps`, and `7g.40gb`. This indicates that MPS and MIG have negligible overhead. On the other hand, we see the impact of having fewer resources available on the `4g.20gb` MIG instance as the workloads get larger in Figures 3a-4a.

Going over to the collocated runs, comparing across the different collocation mechanisms on Figures 2a-4a reveals that MIG-based collocation performs better as the degree of parallelism increases (especially to 7). MPS reveals itself as a clear winner, offering the best performance across the board. In contrast, naïve collocation is the least effective. We attribute the superior performance of MPS to its more flexible resource management allowing more effective collocation (as Section 3.3 also shows) and the lower compute resources that are available to MIG (Section 2.1).

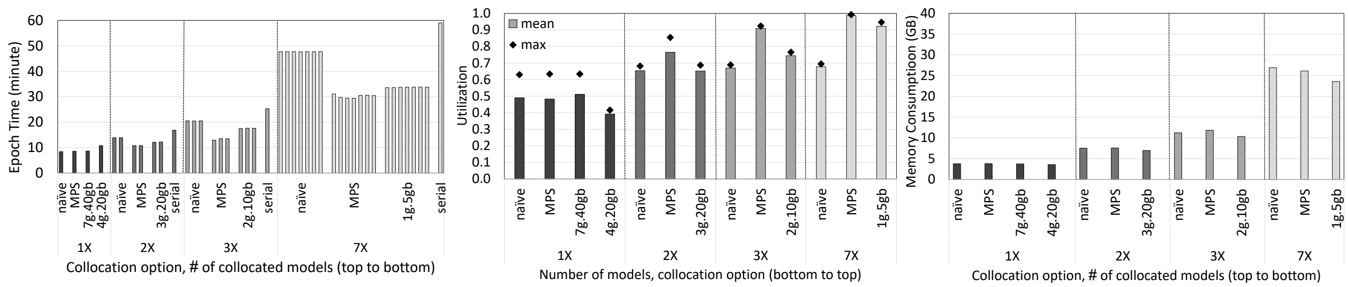
As expected, collocation impacts the time it takes to train the individual models due to interference across the collocated runs. Additionally, as the degree of collocation increases, so does the total time to train the models. On the other hand, multiple models finish training simultaneously, increasing training throughput. For example, except for the large workloads, 2-way collocation delivers two models in roughly the same time as no-collocation delivers one model. 3-way collocation with MPS and MIG leads to a 50-110%

[2] A larger set of results can be found in our longer report [24].



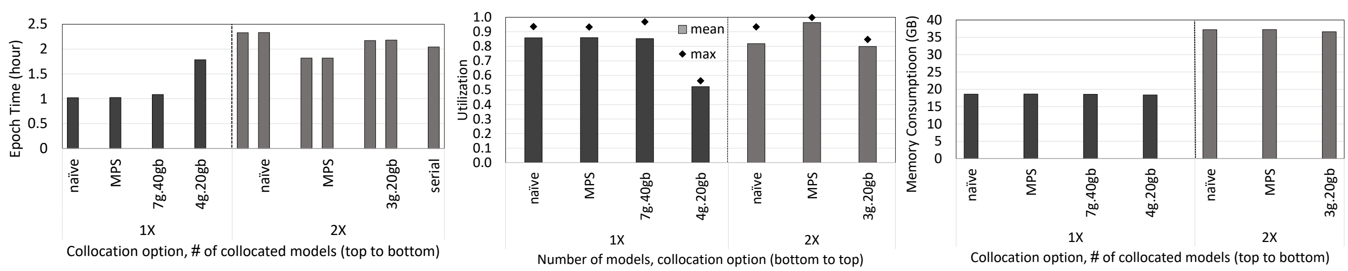
(a) Epoch time **(b) GPU utilization** **(c) Memory footprint**

Figure 2: Small: ResNet26 + Cifar10 (batch size = 128).



(a) Epoch time **(b) GPU utilization** **(c) Memory footprint**

Figure 3: Medium: EfficientNet_s + ImageNet64 (batch size = 128).



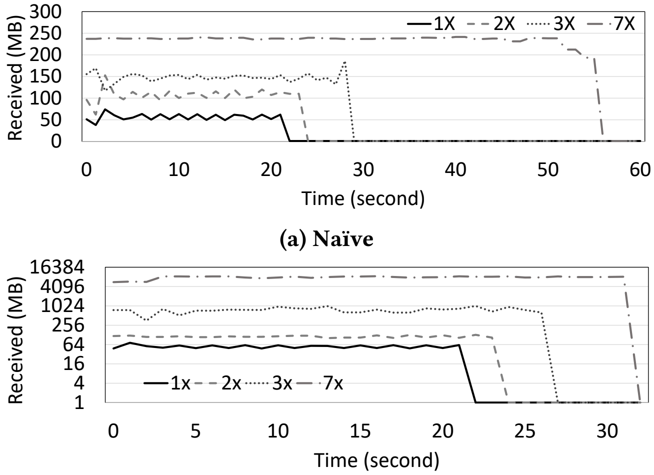
(a) Epoch time **(b) GPU utilization** **(c) Memory footprint**

Figure 4: Large: CaiT + ImageNet (batch size = 128).

increase in time per epoch compared to non-collocated case while delivering three model training runs instead of one. 7-way collocation with MPS and MIG only increases the runtime 2X-2.5X for our smallest workload (Figure 2) while delivering 7 models in parallel. These results clearly show that collocation is valuable when a single training run is not large enough for the available GPU compute and memory resources; e.g., the *small* and *medium* cases.

However, the picture shifts considerably with the large workloads (Figure 4). We no longer see improvements for all of the collocated runs. MPS remains strong and is the only form of collocation that remains beneficial in terms of throughput. Under naïve collocation, one epoch of training takes roughly as long as training the models in sequence without collocation. MIG fairs a little better under 2-way collocation, but is not advantageous. Additionally, 3-way and 7-way collocation becomes impossible due to memory constraints.

3.2.2 GPU utilization. We use SM Activity to track ***GPU utilization***, [35], which is reported by the dcgm tool [19]. For the small case and 7-way collocation, the benefits of collocation become very visible. With ResNet's embarrassingly parallel nature and the larger batch size allowing even more parallelism, high utilization of the GPU compute resources is achieved without overloading the GPU (Figure 2b). The



(a) Naïve

(b) Multi-Process Service

Figure 5: Traffic from CPU to GPU during the second epoch of ResNet26 + Cifar10 (batch size 32) training.

medium case reflects the same pattern, though starts hitting compute resource boundaries under 7-way collocation, as seen in Figure 3b. As a result, collocation provides considerable benefits for the small and medium cases with MIG and especially with MPS. For the large case (Figure 4b), there is little variety in the GPU utilization across different cases.

3.2.3 GPU memory footprint. Finally, Figures 2c-4c report the aggregate ***memory footprint*** on the GPU for different collocation methods for each workload. We use `nvidia-smi` to collect the memory consumption for the whole GPU after a full epoch of training to signify how much memory is needed for the model to train. The figures demonstrate that the increase in memory footprint with collocation is proportional to the degree of collocation. This is an expected result as the models are not sharing data across collocated runs in these experiments.

Notably, MIG collocation shows slightly smaller memory footprints than the two other options, which prompted us to delve deeper into PyTorch's memory allocation. We discovered that PyTorch adjusts the memory footprint depending on the total available memory, which is less in the case of non-`7g.40gb` MIG instances compared to whole GPU memory available under MPS and naïve. Switching the memory allocator backend from PyTorch's native implementation to CUDA's built-in asynchronous allocator removes the differences in the memory footprint of different collocation methods. However, we do not recommend this switch as it slows down the training process.

3.2.4 Interconnect Traffic. Figure 5 reports the amount of bytes received over time by the GPU measured by `dcgm`'s

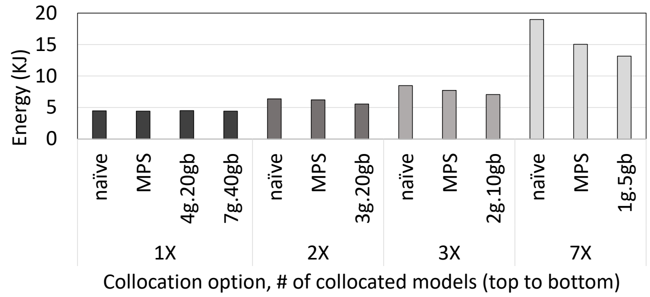
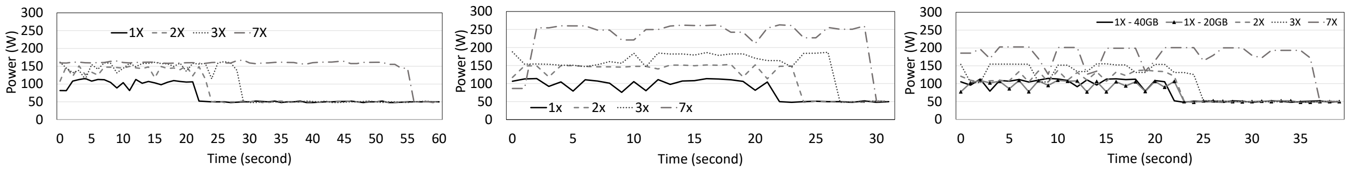


Figure 6: GPU energy consumption to complete the 2nd epoch of ResNet26 + Cifar10 (batch size 32) training.

`pcie_rx_bytes`. We compare naïve and MPS collocation during the second epoch of small ResNet training with batch size 32. We pick this small case as it benefits greatly from collocation and can highlight the differences across the collocation scenarios more effectively. MIG is omitted here due to `dcgm` not providing the readings for this metric under MIG as a result of the GPU being split into multiple instances.

For lower degrees of collocation, naïve collocation leads to a linear increase in data transferred over PCIe from CPU to GPU with respect to degree of collocation. On the other hand, for the 7-way case, there is less work being done per unit of time for each training run leading to sub-linear PCIe traffic. This is likely caused by the throughput benefits of collocation taking a huge hit under naïve collocation, as shown in Section 3.2.1. In contrast, MPS exhibits a super-linear increase in PCIe utilization when collocating models. In addition to the data transfers for the collocated runs, MPS increases the kernel launch processes since it is able to eliminate false dependencies and share the GPU resources more effectively across the collocated kernels (Section 2.1).

3.2.5 Energy Consumption. Finally, we look at the power usage and GPU energy consumption using `dcgm`'s `power_usage` (watts) and `total_energy_consumption` (joules), respectively, for the small ResNet training. Figure 7 shows that collocation scenarios that are highly effective may run on higher power but finish much quicker. This is due to higher GPU utilization under MPS and MIG. MIG exhibits significantly lower wattage under 7-way collocation than MPS while training slightly slower. The benefits of this can be seen in Figure 6, which reports the total GPU energy consumption of the second epoch of the model training. While requiring higher power usage per unit of time, MPS spends less energy compared to naïve collocation thanks to finishing training faster. While not as fast as MPS, MIG in general exhibits the lowest GPU energy footprint.



(a) Naïve

(b) Multi-Process Service

(c) Multi-Instance GPU

Figure 7: GPU power usage during the second epoch of ResNet26 + Cifar10 (batch size 32) training.

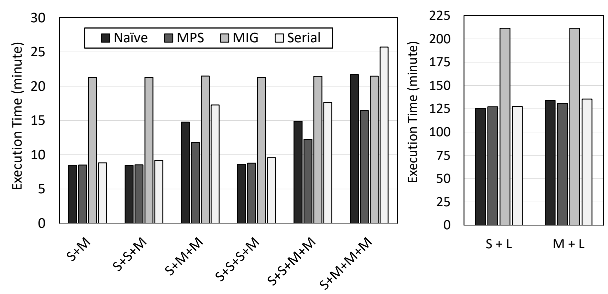


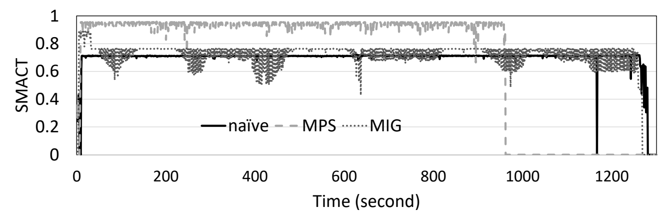
Figure 8: Time for training mixed vision workloads with & without (serial) collocation for two epochs.

3.3 Mixed Workloads

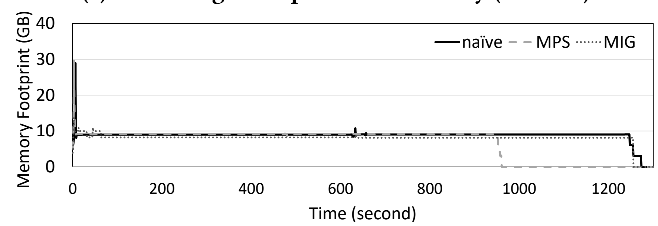
So far, we focused on homogeneous collocation scenarios. Such cases can be extremely useful in practice when a data scientist is performing hyper-parameter tuning to come up with the ideal set of parameters for a model repeatedly running the same model with a different set of parameters. On the other hand, there is also value in investigating non-homogeneous collocation scenarios to observe what happens when individual training runs stress the GPU unequally.

We select combinations of small, medium, and large ResNet models with corresponding dataset sizes to collocate for the heterogeneous runs (as listed in Table 1). For the MIG workloads, these run on `1g.5gb`, `2g.10gb` and `4g.20gb`, respectively. We opted to keep a static MIG configuration while testing heterogeneous collocation since in a real-world scenario, e.g., in a data center, the MIG partitions would already be set and reallocating resources after each training run could be impractical.

Figure 8 details the total execution time for training the collocated models using the different collocation methods in comparison to training them back to back, *serial*, without collocation. We see that the benefits of collocation vary heavily across workloads. For larger workloads such as "S+M+M" and "S+S+M+M", naïve and MPS collocation provide sizeable benefits by training the small model without impacting the medium one. In general, the flexibility of both naïve collocation and MPS is a great advantage here over MIG.



(a) Streaming Multiprocessor Activity (SMACT)



(b) Memory footprint

Figure 9: GPU utilization and memory footprint over time for S+M+M+M from Figure 8.

Figure 9 dives deeper into the "S+M+M+M" workload to observe how the GPU utilization and memory footprint changes over time during collocated runs with naïve, MPS, and MIG collocation. We pick this mix as it is the one that utilizes MIG instances the best. The GPU utilization under MIG gets lowered after the small model finishes, since MIG is unable to fill-up the corresponding instance with more work. On the other hand, naïve and MPS are able to keep similar GPU utilization throughout. In contrast, the memory footprint follows a similar trend for all collocation strategies. It is higher in the beginning as all four models are training. The values then drop off quickly once the small model finishes training.

Furthermore, to investigate the impact of collocating mixed workloads that stress different hardware resources, we show the results of collocating a recommender model with a large vision model training in Table 2. We configure two 3g MIG compute instances to share memory as the recommender model does not fit into the memory of smaller GPU instances.

Adding a memory-heavy model such as the recommender greatly promotes collocation. Training time only increases slightly when collocating these, going between 4%-14% and

Table 2: Mixed collocation of memory-intensive recommender and compute-intensive vision models. Recommender time is for one training block plus validation. ResNet time is for one epoch. The reported increase in time (%) is relative to the sequential run.

Workload	Time (h)			GPU Util.	Memory (GB)
	Recom.	ResNet	**Total**		
Recommender	5.36	-	**6.41**	5%	29.14
ResNet152	-	1.05		82%	8.47
Naïve	6.09 (+14%)	1.11 (+6%)	**6.09 (-5%)**	81%	37.75
MPS	5.57 (+4%)	1.10 (+5%)	**5.57 (-13%)**	81%	37.62
MIG (shared)	5.60 (+4%)	1.40 (+33%)	**5.60 (-13%)**	39%	37.86

4%-33% for the recommender and ResNet, respectively. MPS performs especially well, training both models with just a 4% increase in training time. As before, memory consumption roughly corresponds to the sum of both models. However, GPU utilization does not increase. Under MIG, unfortunately, only part of the computing power of the GPU can be assigned to ResNet, even though the recommender requires little.

3.4 Summary & Collocation Guidelines

Based on the results we covered, we now provide some guidelines for deep learning training collocation.

- Workload collocation is highly beneficial when the aggregate compute and memory needs of the collocated deep learning training runs fit the GPU.
- Collocation gives diminishing returns when the GPU utilization of an individual run is already close to 100%.
- The aggregate memory footprint of the collocated runs can effectively be estimated by the sum of the memory footprints of the individual runs and cannot exceed the available memory on the GPU.
- MPS achieves better performance across the board thanks to its flexible distribution of hardware resources among the collocated runs. On the other hand, it requires higher interconnect bandwidth.
- MIG can support collocation effectively when a strict separation is required among the runs thanks to its rigid partitioning even though this partitioning leads to suboptimal performance compared to MPS. Furthermore, MIG exhibits higher energy efficiency on GPUs when the instances are configured well for the workload.

4 CONCLUSION

In this paper, we provide a performance characterization on a modern GPU device that has support for multiple means of GPU collocation: naïve, MPS, and MIG. Our results demonstrate that GPU collocation is highly beneficial for small- and medium-sized workloads that cannot fully saturate the whole GPU. Although per-model training is overall slower, parallel execution of workloads can utilize GPU resources more effectively, increasing training throughput. MIG notably requires a rigid setup while providing full isolation across its instances.

If the workload across the instances is imbalanced, runs that finish early will leave some instances idle, unless there is other work that could be allocated over those instances. Naïve collocation and MPS, on the other hand, can utilize the resources released by the finished work, increasing the training performance of models that require more time to train. In general, MPS provides the best collocation performance, if not the most energy efficient.

In this work, we limited our focus to training on a single GPU since NVIDIA does not allow multi-GPU training with MIG. we limited our focus to training on a single GPU since NVIDIA does not allow multi-GPU training with MIG. In a data center, many workloads can be collocated not only on the same GPU but also on the same server. Therefore, studying the impact of collocation while running other workloads on other GPUs on the same device would be interesting future work. Furthermore, considering the results with the recommender model, further investigations of the shared memory instances of MIG would be worthwhile.

ACKNOWLEDGEMENTS

This work is funded by the Independent Research Fund Denmark's (Danmarks Frie Forskningsfond; DFF) Sapere Aude program under grant agreement number 0171-00061B and Inge Lehman program under grant agreement number 0171-00062B. We also thank DASYA lab members at IT University of Copenhagen for their support, and the reviewers of EuroMLSys for their constructive feedback.

REFERENCES

[1] [n.d.]. GPU Pro Tip: CUDA 7 Streams Simplify Concurrency. https://developer.nvidia.com/blog/gpu-pro-tip-cuda-7-streams-simplify-concurrency/. Accessed: 2022-10-21.

[2] Sebastian Baunsgaard, Sebastian Benjamin Wrede, and Pınar Tözün. 2020. Training for Speech Recognition on Coprocessors. In *ADMS*.

[3] Mehmet E. Belviranli, Farzad Khorasani, Laxmi N. Bhuyan, and Rajiv Gupta. 2016. CuMAS: Data Transfer Aware Multi-Application Scheduling for Shared GPUs. In *Proceedings of the 2016 International Conference on Supercomputing (ICS '16)*. Association for Computing Machinery, Article 31, 12 pages.

[4] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. 2017. A Downsampled Variant of ImageNet as an Alternative to the CIFAR datasets. *CoRR arXiv* (2017).

[5] Criteo. [n.d.]. Criteo 1TB Click Logs dataset. https://www.criteo.com/news/press-releases/2015/07/criteo-releases-industrys-largest-ever-dataset/.

[6] Hongwen Dai, Zhen Lin, Chao Li, Chen Zhao, Fei Wang, Nanning Zheng, and Huiyang Zhou. 2018. Accelerate GPU Concurrent Kernel Execution by Mitigating Memory Pipeline Stalls. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 208–220.

[7] Sina Darabi, Negin Mahani, Hazhir Baxishi, Ehsan Yousefzadeh-Asl-Miandoab, Mohammad Sadrosadati, and Hamid Sarbazi-Azad. 2022. NURA: A Framework for Supporting Non-Uniform Resource Accesses in GPUs. *Proc. ACM Meas. Anal. Comput. Syst.* 6, 1 (feb 2022).

[8] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-Efficient and QoS-Aware Cluster Management. In *ASPLOS*. 127–144.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR*. 770–778.

[10] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. 2011. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In *NSDI*. 295–308.

[11] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. 2019. Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. 947–960.

[12] Alexandros Koliousis, Pijika Watcharapichat, Matthias Weidlich, Luo Mai, Paolo Costa, and Peter Pietzuch. 2019. Crossbow: Scaling Deep Learning with Small Batch Sizes on Multi-GPU Servers. *PVLDB* 12, 11 (2019), 1399–1412.

[13] Alex Krizhevsky. 2009. *Learning multiple layers of features from tiny images.* Technical Report. University of Toronto.

[14] Baolin Li, Viiay Gadepally, Siddharth Samsi, and Devesh Tiwari. 2022. Characterizing Multi-Instance GPU for Machine Learning Workloads. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 724–731.

[15] Baolin Li, Tirthak Patel, Siddarth Samsi, Vijay Gadepally, and Devesh Tiwari. 2022. MISO: Exploiting Multi-Instance GPU Capability on Multi-Tenant GPU Clusters. In *ACM SoCC*.

[16] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G. Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong, and Misha Smelyanskiy. 2019. Deep Learning Recommendation Model for Personalization and Recommendation Systems. *CoRR* abs/1906.00091 (2019). https://arxiv.org/abs/1906.00091

[17] Konstantinos Nikas, Nikela Papadopoulou, Dimitra Giantsidi, Vasileios Karakostas, Georgios Goumas, and Nectarios Koziris. 2019. DICER: Diligent Cache Partitioning for Efficient Workload Consolidation. In *ICPP*.

[18] NVIDIA 2021. *NVIDIA Multi-Instance GPU User Guide.* NVIDIA. https://docs.nvidia.com/datacenter/tesla/mig-user-guide/.

[19] NVIDIA. 2022. *Data Center GPU Manager Documentation.* Technical Report. NVIDIA. https://docs.nvidia.com/datacenter/dcgm/latest/dcgm-user-guide/.

[20] NVIDIA. 2022. *Multi-Process Service.* Technical Report. NVIDIA Corporation. https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf

[21] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. 8024–8035.

[22] Vignesh T. Ravi, Michela Becchi, Gagan Agrawal, and Srimat Chakradhar. 2011. Supporting GPU Sharing in Cloud Environments with a Transparent Runtime Consolidation Framework *(HPDC '11)*. Association for Computing Machinery, 217–228.

[23] Ties Robroek, Aaron Duane, Ehsan Yousefzadeh-Asl-Miandoab, and Pinar Tözün. 2023. Data Management and Visualization for Benchmarking Deep Learning Training Systems. In *Proceedings of the Seventh Workshop on Data Management for End-to-End Machine Learning, DEEM 2023, Seattle, WA, USA, 18 June 2023*. ACM, 1:1–1:5. https://doi.org/10.1145/3595360.3595851

[24] Ties Robroek, Ehsan Yousefzadeh-Asl-Miandoab, and Pınar Tözün. 2023. An Analysis of Collocation on GPUs for Deep Learning Training. arXiv:2209.06018 [cs.LG]

[25] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *IJCV* 115, 3 (2015), 211–252.

[26] Mingxing Tan and Quoc V. Le. 2021. EfficientNetV2: Smaller Models and Faster Training. *CoRR* abs/2104.00298 (2021). arXiv:2104.00298 https://arxiv.org/abs/2104.00298

[27] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. 2021. Going deeper with image transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 32–42.

[28] Shang Wang, Peiming Yang, Yuxuan Zheng, Xin Li, and Gennady Pekhimenko. 2021. Horizontally Fused Training Array: An Effective Hardware Utilization Squeezer for Training Novel Deep Learning Models. *Proceedings of Machine Learning and Systems* 3 (2021), 599–623.

[29] Zhenning Wang, Jun Yang, Rami Melhem, Bruce Childers, Youtao Zhang, and Minyi Guo. 2016. Simultaneous Multikernel GPU: Multi-tasking throughput processors via fine-grained sharing. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 358–369.

[30] Zhenning Wang, Jun Yang, Rami Melhem, Bruce Childers, Youtao Zhang, and Minyi Guo. 2017. Quality of service support for fine-grained sharing on GPUs. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. 269–281.

[31] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. 2022. MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, 945–960. https://www.usenix.org/conference/nsdi22/presentation/weng

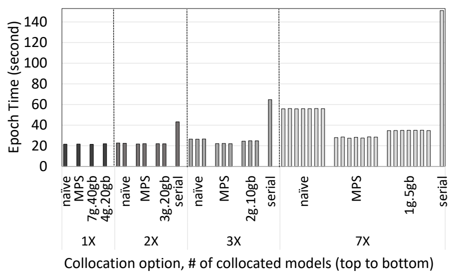
[32] Ross Wightman. 2019. PyTorch Image Models. https://github.com/rwightman/pytorch-image-models.

[33] Qiumin Xu, Hyeran Jeon, Keunsoo Kim, Won Woo Ro, and Murali Annavaram. 2016. Warped-Slicer: Efficient Intra-SM Slicing through Dynamic Resource Partitioning for GPU Multiprogramming. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 230–242.

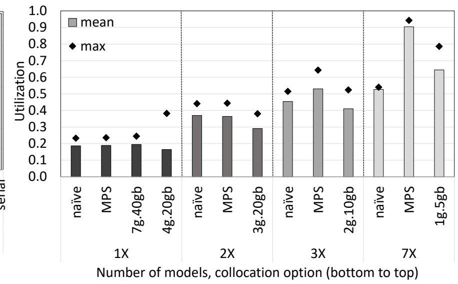
[34] Gingfung Yeung, Damian Borowiec, Renyu Yang, Adrian Friday, Richard Harper, and Peter Garraghan. 2022. Horus: Interference-Aware and Prediction-Based Scheduling in Deep Learning Systems. *IEEE Transactions on Parallel and Distributed Systems* 33, 1 (2022), 88–100. https://doi.org/10.1109/TPDS.2021.3079202

[35] Ehsan Yousefzadeh-Asl-Miandoab, Ties Robroek, and Pinar Tözün. 2023. Profiling and Monitoring Deep Learning Training Tasks. In *Proceedings of the 3rd Workshop on Machine Learning and Systems, EuroMLSys 2023, Rome, Italy, 8 May 2023*, Eiko Yoneki and Luigi Nardi (Eds.). ACM, 18–25. https://doi.org/10.1145/3578356.3592589

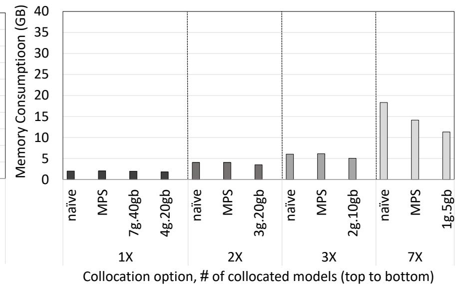
[36] Xia Zhao, Zhiying Wang, and Lieven Eeckhout. 2018. Classification-Driven Search for Effective SM Partitioning in Multitasking GPUs. In *Proceedings of the 2018 International Conference on Supercomputing*. 65–75.



(a) Epoch time

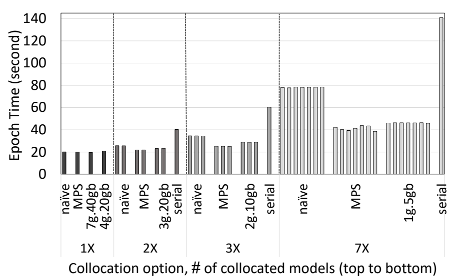


(b) GPU utilization

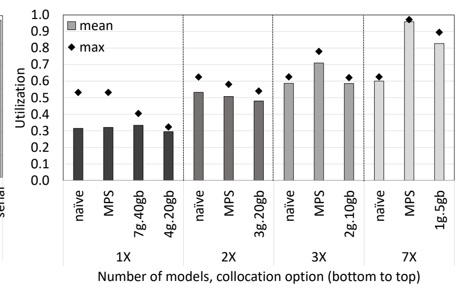


(c) Memory footprint

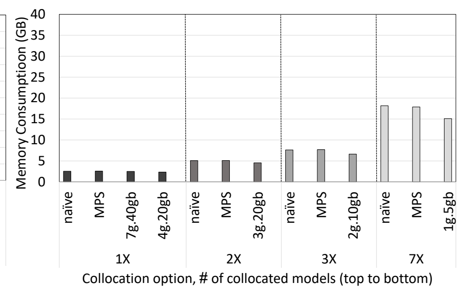
Figure 10: Small: ResNet26 + Cifar10 (batch size = 32).



(a) Epoch time

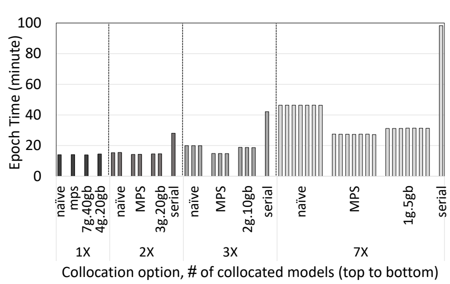


(b) GPU utilization

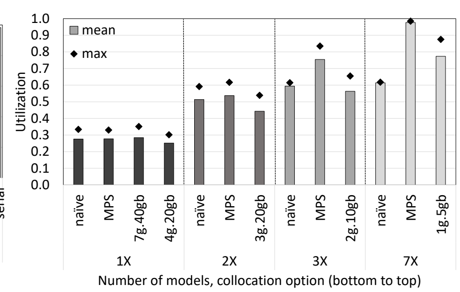


(c) Memory footprint

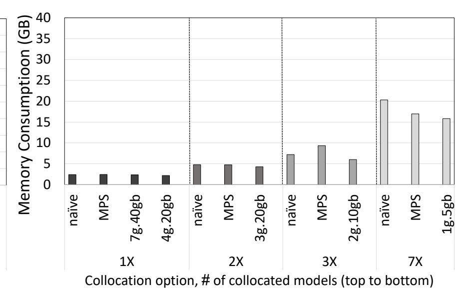
Figure 11: Small: EfficientNet_s + Cifar10 (batch size = 128).



(a) Epoch time



(b) GPU utilization

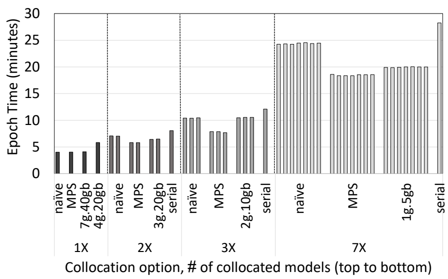


(c) Memory footprint

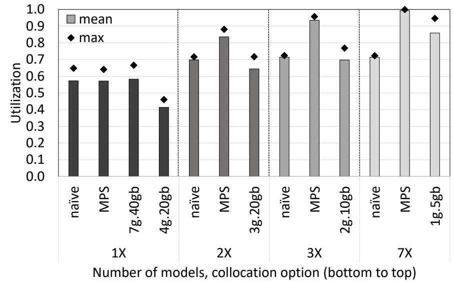
Figure 12: Medium: ResNet50 + ImageNet64 (batch size = 32).

A ADDITIONAL UNIFORM COLLOCATION RESULTS

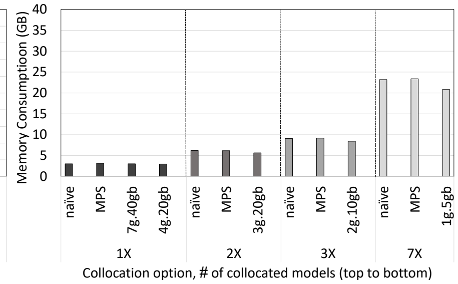
As part of our investigation of the collocation mechanisms, we have also experimented with varying the batch size and tested out additional model and dataset combinations. We are sharing the results from those experiments in this appendix for completeness in Figures 10-14, even though they do not change the key conclusions of this paper.



(a) Epoch time

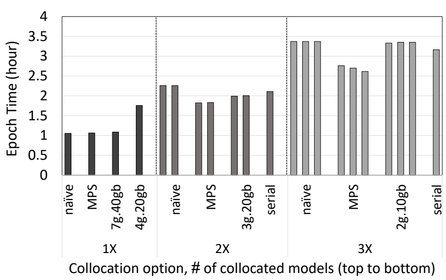


(b) GPU utilization

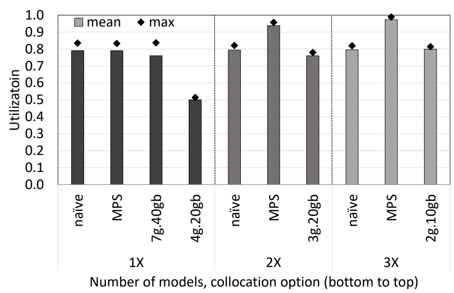


(c) Memory footprint

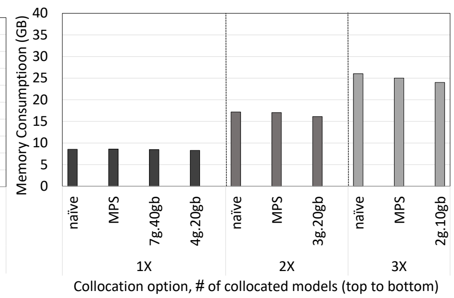
Figure 13: Medium: ResNet50 + ImageNet64 (batch size = 128).



(a) Epoch time



(b) GPU utilization



(c) Memory footprint

Figure 14: Large: ResNet152 + ImageNet (batch size = 32).