Brilliance
Research of Artificial Intelligence

E-ISSN : 2807-9035
Volume 5, **Number** 2, November 2025
https://doi.org/10.47709/brilliance.v5i2.6794

Benchmarking GPU Passthrough Performance on Docker for AI Cloud System

Ahmad Faisal Sani[1*], Rifa Khoirunisa[2], Darmawan Lahru Riatma[3], Yusuf Fadlila Rachman[4], Masbahah[5]
[1,2,3,4,5] Universitas Sebelas Maret K.Kab. Madiun, Indonesia
[1]faisalsani@staff.uns.ac.id, [2]rkhoirunisa@staff.uns.ac.id, [3]darmawanlr@staff.uns.ac.id, [4]yusuf_fadil@staff.uns.ac.id, [5]masbahah@staff.uns.ac.id

open access

***Corresponding Author**

Article History:
Submitted: 31-07-2025
Accepted: 09-08-2025
Published: 12-08-2025

Keywords:
Artificial Intelligence; Docker Container; GPU; GPU Passthrough; NVIDIA Cuda.

ABSTRACT

The use of artificial intelligence (AI), which depends only on CPU resources, tends to result in longer execution times or CPU time. Especially when handling large amounts or complex workloads. To overcome that issue, the use of a graphics processing unit (GPU) becomes a significant support. GPUs can significantly speed up AI inferences through their parallel architecture. One recent approach to integrating GPUs into an AI system is called GPU passthrough. Either natively (native environment), or through Docker environment. However, until recently, the efficiency and results between those methods have remained unexplored, particularly in local cloud environment.This study aimed to compare GPU performance between native and Docker environment using a 10.000 x 10.000 matrix multiplication workload with the TensorFlow frameworks. Execution time and GPU performance measured using the nvidia-smi tool. Data is recorded automatically in CSV format. The researcher used the NVIDIA CUDA environment to ensure full compatibility with GPU acceleration.The result demonstrated that GPU processing in native environment had faster average time, as in 1.52 seconds. In another case, GPU passthrough in docker environment demonstrated higher GPU utilization, as in 86.2% but had a longer execution time.These findings indicate that GPU overhead occurred in docker environment due to the containerization layer. On the contrary, the native environment resulted in shorter execution time, even though it did not maximize the GPU utilization. These results provide valuable basis data for technical decision-making in GPU-based AI deployment in a limited environment.

INTRODUCTION

Nowadays, Artificial Intelligence (AI) has becomes main innovation in various sectors. Including healthcare, agriculture, finance, and transportation. To fully maximize true potential of AI, requires large-scale, and flexible computational resources.

Actually, AI inference process, such as; image classification, object detection, and natural language processing, require the support of graphics processing units (GPUs). In addition, both AI training and inference are intensive computational processes. In order all of those processes to run smoothly and fast, is recommended to use GPUs rather than only CPU (Walters et al., n.d.).

Better understand the context of this research, it is important to first explore the role of Graphics Processing Units (GPUs) in greater details. GPUs are hardware made to perform graphical rendering and numerical computations in parallel. To overcome all of those tasks, GPUs come with thousands of small cores(Kurkure et al., 2017). GPUs are capable of executing many operations simultaneously, so it is very efficient to solve large-scale data processing.

GPUs are particularly well-suited for machine learning and AI applications(Chang et al., 2020). GPUs role in AI is critical, due to processes such as image classification, and natural language processing, demand intensive computational power. GPUs are able not only to accelerate AI model training, but also reduce the inference time, making faster development and deployment of AI solutions.

NVIDIA GPUs has programming layer called NVIDIA CUDA (Compute Unified Device Architecture). CUDA is a parallel computing and programing model platform which enabled developer to utilize GPU for general-purpose tasks. CUDA is also enabled programmers to code in various popular programming languages such as C, C++, and Fortran, which is then executed in parallel by thousands of cores within the GPUs. This paralelism significantly accelerates the processing of large and complex datasets, as commonly appeared in AI applications.

Cuda is also facilitates various applications, from graphic computer, scientific simulations, and, deep learning. It makes GPUs vital component in the development of modern technologies that leverage the high computing data (Belkhiri & Dagenais, 2024). Main advantages of using CUDA lies in its ease of use and its ability to directly manipulate GPU memory. That feature is allowing developers to optimize application performance efficiently.

Brilliance
Research of Artificial Intelligence

E-ISSN : 2807-9035
Volume 5, **Number** 2, November 2025
https://doi.org/10.47709/brilliance.v5i2.6794

Furthermore, CUDA is also have a set of highly optimized libraries designed to enhance performance in numerical computing and data analysis. As results of that, CUDA accelerating critical function in machine learning and AI (Choi et al., 2021). These features make CUDA a fundamental tool in the technology industry, where the demand for fast and efficient computation increasingly dominates various domains and applications.

Native GPU usage can be inefficient as a single GPU typically runs only one AI app at a time. To improve resource utilization, GPU Passthrough is employed in virtualization environments such as Proxmox (Zhao et al., 2025). In Proxmox, virtual GPU Passthrough (vGPU) can be implemented by enabling a single NVIDIA GPU to be partitioned and shared across multiple virtual machines.

However, the vGPU Passthrough method is typically less practical for local environments or educational purposes. This is primarily because GPU Passthrough lacks support for consumer-grade NVIDIA graphic cards, such as the RTX 3000, 4000, and 5000 series, which are equipped with CUDA and Tensor Cores.

In addition, Virtual Machines (VMs) strongly rely on an additional layer called hypervisor, which runs on top of the host operating system. The function is to emulate hardware to produce virtual machine. However, the addition of hypervisor will decrease the performance of the virtual machine itself.

To overcome this weakness, the researcher uses an alternative which claim to be lighter than virtual machines. This alternatives called as Container (Kumar & Kaur, 2022). Over time, Container become more popular because of its advantages, such as; lower resource consumption, and easy to make (Oh et al., 2019).

In this study, the researcher will apply Container technique called Docker. Container Docker can also be called as light and efficient virtualization technology(Shea & Liu, n.d.). This feature enables developers to contain the application and all of its dependences in one unit which run together consistenly in all of computing environment.

A Docker container uses the kernel from the host operating system and isolates each container with app and services needed. It creates process which low consuming resource and time efficient (Openja et al., 2022). Docker also support fast-paced app distribution, which simplify development and management of cloud-based solution. Docker container also increase the efficiency of machine learning and web-based application . Docker container even more special because it replaces the necessity of heavy virtual machine. We can say goodbye to overhead and welcome the increasing performance of running apps simultaneously (Shetty et al., 2017).

To assist Docker Container, the researcher uses NVIDIA Container Toolkit which enable Docker accesses NVIDIA GPU directly to run CUDA-based app (Tiying & Zhengwei, 2019) such as deep learning, computer vision, and data analysis. This toolkit includes special runtime (nvidia-container run time), and library (libnvidia-container) which automatically mapping the GPU and CUDA library from host to container (NVIDIA, n.d.-a). By only adding the flag --gpus all, the container can utilize the GPU as it would on a native system. The toolkit is also support various AI framework, such as TensorFlow and Pytorch. Moreover, the toolkit is compatible with modern docker and environmental orchestration like Kubernetes.

Later on, the researcher compares the GPU performance in native environment, and the GPU performance via GPU passthrough using Docker Container. Then the researcher classifies the efficiency and the overhead between those two environments.

This study is important because provides empirical and practical insight about the efficiency of workload execution on GPU-based AI in local environments, an area that has not been studied much before. This study also support in decision making technique between the using of container and native execution.

The purpose of this study is to compare the performance and utilization of GPU between native environment and Container Environment using Docker. The researcher limits the work field in local cloud. Hopefully this research can provide technical recommendations to practitioner and academics in order to choose the most suitable methods to maximize the AI inference with local GPU.

LITERATURE REVIEW

In the last couple years, cloud computing technology and containerization has grown exponentially. It is reflected in the using of GPU in order to produce Virtual Machine (VM) on OpenStack and Docker. This research explores the application of GPU on those two environments and the effect on performance and efficiency data processing.

First, GPU plays an important role in leveling up the application performance, especially in machine learning and graphic processing. Belkhiri and Dagenais wrote that GPU virtualization provides scalable GPU resource which adaptive to virtual machine. This features increases apps efficiency and responsiveness, which run on top of it (Belkhiri & Dagenais, 2024).

In the case of OpenStack, Zhao et al. explains that the using of GPU Passtrough allows direct resource allocation to the host cloud. This method significantly fulfills the necessity of high level computing data (Zhao et al., 2025). This stats show that GPU integration in cloud environment amplify speed and data efficiency process.

Furthermore, the implementation of Docker as container platform enables high flexibility in management and deployment of application that need GPU. Based on Shetty et al., Docker offers performance level that similar with native. It is better that virtualization technology based on hypervisor like KVM. All of these advantages are possible by its architecture which need no additional abstraction layer between the hardware and the virtual machine (Shetty et al.,

Brilliance
Research of Artificial Intelligence

E-ISSN : 2807-9035
Volume 5, **Number** 2, November 2025
https://doi.org/10.47709/brilliance.v5i2.6794

2017).

When used with GPUs, Docker allows developers to package all the application with all dependencies needed. The feature makes it easier to deploy GPU oriented application in the cloud (Openja et al., 2022). It makes higher rate efficiency possible and better resource management. This study also integrate container with OpenStack to create better scheduling technique. The technique is particularly useful for maximizing GPU resource utilization (Lingayat et al., 2018). Through the research by Chang et al., shows that increasing GPU efficiency resource can be reach by using virtually accelerated GPU (Chang et al., 2020). This approach is very crucial in the developing cloud-computing era.

Besides all the advantages above, the use of GPU on Docker and OpenStack also quiet challenging. The primary challenges are the management and maintenance so the hardware performance keeps optimum. User must regularly maintain network traffic and manage the resource in multi-tenant environment. Those challenges can be complicated if not manage properly (Čisar et al., 2018).

The combination of GPU, Docker, and OpenStack enables new opportunity in cloud computing development. The use of this technology – from machine learning to graphic rendering- offer significant advantages, especially for organization or team who want to utilize cloud-based infrastructure.

Despite the fact that various studies have examined the use of GPU with virtual machine and containers, most of it have focused on large-scale cloud environments such as OpenStack and Kubernetes. It is utilizing enterprise GPU configuration and centralized pools(Belkhiri & Dagenais, 2024; Zhao et al., 2025). Such approaches are often not applicable to individual or organization with limited funding which can only rely on consumer-grade GPUs.

Moreover, prior research has largely emphasized deployment capability and compatibility. It has not yet been widely discussed about quantitative analysis on GPU performance using Docker vs. Native. Similar studies by Shetty et al. (2017) and Openja et al. (2022) are also comparing a GPU container without a direct experiment to compare the utilization and the efficiency of the GPU.

METHOD

This study uses a quantitative experiment to measure and compare the execution performance and utilization of the GPU in two modes. Those are workload AI executions in the native environment and Docker container. Main workload uses are big matrix multiplication of 10,000x10,000, which generally represents computing workload on AI inferences.

The use of matrix multiplication as a workload benchmark is regarded as standard practice in AI computing research (Shi et al., 2017; Wang et al., 2019). It reflects the fundamental Neural Network. GPU monitoring with the nvidia-smi tool is chosen because the accuracy is high and represents the industry (NVIDIA, n.d.-b).

Experimental Environment

The experiments are conducted in a local system with specifications as shown below:
GPU: NVIDIA RTX 3060 (12 GB VRAM)
CPU: Intel Core i5-12400
RAM: 32 GB DDR4
OS Host: Ubuntu 22.04 LTS
CUDA Toolkit: Version 12.8
Driver NVIDIA: Versi 570.133.07

Experimental Stages:

The experimental process consisted of four main stages. First, the environmental setup was conducted by installing compatible NVIDIA drivers, the CUDA toolkit, and the NVIDIA Container Toolkit to ensure GPU support in both native and containerized (Docker) environments. Second, in the workload preparation stage, a script named infer.py was developed using TensorFlow to perform large-scale matrix multiplication. This script was executed ten times in each environment—native and Docker—to ensure consistency and comparability of results.

Third, the measurement stage involved two key aspects: execution time and GPU utilization. Execution time was measured using the time.time() function within the Python script to capture the duration of each run. Simultaneously, GPU utilization was monitored using the nvidia-smi tool in polling mode, which logged GPU usage every 0.1 seconds during execution.

Finally, in the logging and analysis stage, all data—including execution time and GPU utilization—were recorded automatically in CSV format using a shell script. These datasets were then subjected to basic statistical analysis to calculate the average execution time and GPU utilization for each execution mode, enabling clear comparison and interpretation of performance differences between native and containerized environments.

Research Instrument

Brilliance
Research of Artificial Intelligence

E-ISSN : 2807-9035
Volume 5, **Number** 2, November 2025
https://doi.org/10.47709/brilliance.v5i2.6794

Table 1. Research Instrument

Instrument	Function
TensorFlow	Running a matrix multiply
NVIDIA-SMI	Monitoring real-time GPU utilization
Bash Script	Automatic logging time and utilization to CSV
Docker CLI	Running a container with GPU access
Python Stopwatch	Measuring the execution duration of an AI workload

RESULT

This study benchmarks the performance of GPU-based TensorFlow workload execution in two environments: native (directly on the host system) and Docker container based on the image in nvcr.io/nvidia/tensorflow:24.05-tf2-py3. The workload used is a large matrix multiplication, representing a standard type of computation in AI model inference.

Table 2. Representing a Standard Type

Run	Time		GPU Utilization	
	Native	Docker	Native	Docker
1	1.5417	2.5334	37	99
2	1.5059	2.5513	69	85
3	1.5214	2.5671	11	61
4	1.5492	2.5134	72	100
5	1.5286	2.5545	24	60
6	1.5333	2.5538	68	95
7	1.5127	2.5327	12	99
8	1.5033	2.5697	33	63
9	1.5028	2.5438	57	100
10	1.4941	2.6024	73	100

The benchmark showed that TensorFlow execution in native is faster compared to Docker. The average is 1.52 seconds compared to 2.55 seconds. The result implies that the native environment is more efficient in handling the task. This is likely due to the absence of containerization overhead and direct access to the hardware resources.

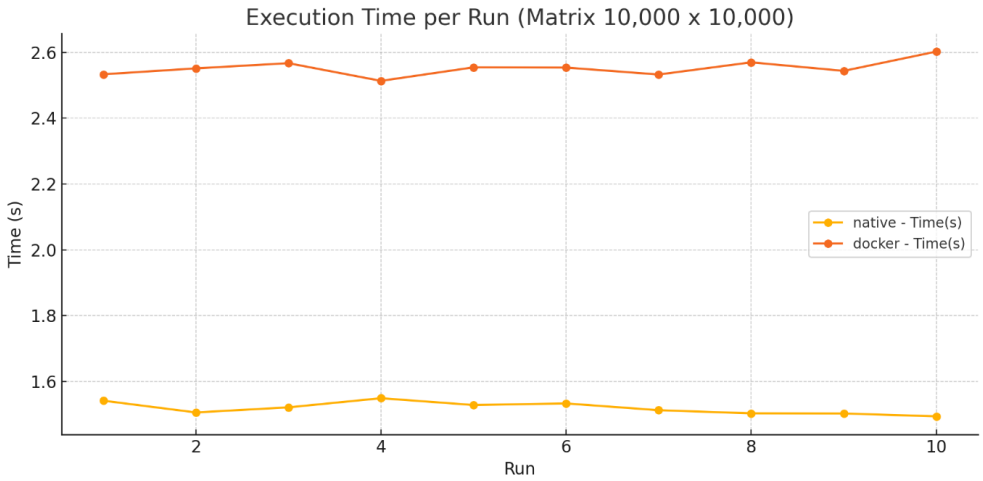

Fig. 1 Comparation execution time per run native and docker

GPU Utility Analysis

Brilliance
Research of Artificial Intelligence

E-ISSN : 2807-9035
Volume 5, **Number** 2, November 2025
https://doi.org/10.47709/brilliance.v5i2.6794

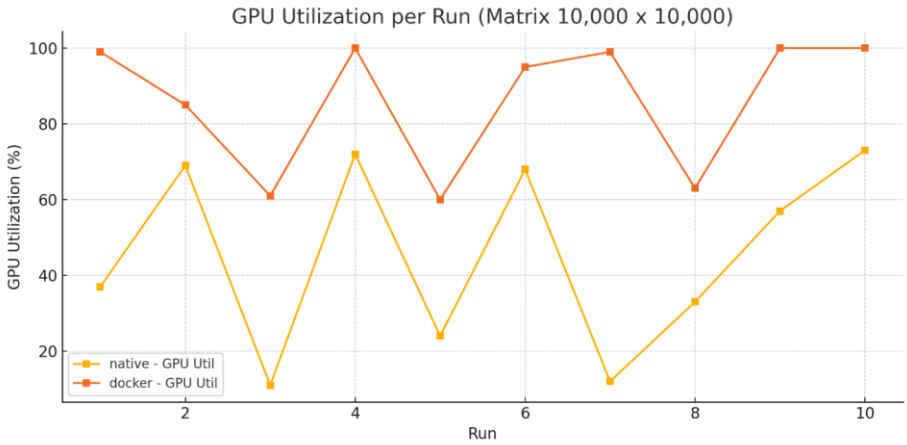

Fig. 2 Comparation GPU Utilization

Docker mode created significantly higher GPU utilization than native mode. The average results are 86%, with several runs reaching 100%. This suggests that the nvcr.io/nvidia/tensorflow image and the Docker environment are highly optimized for GPU workloads. In contrast, the native mode exhibits lower GPU utilization, averaging 45,6%, despite achieving faster execution times. This indicates that the GPU was used more efficiently in the native environment.

Implication and Evaluation

These findings offer valuable insights into the use of a GPU for an AI workload in a local environment. The primary goal is to achieve minimal execution time, and a native environment is preferable. However, in order to maximize GPU utilization and ensure consistent run time, a Docker environment is an advisable alternative.

For educational institutions or research centers with limited resources, these results can serve as a practical reference for designing AI execution pipelines – both for training and inference- on local GPU infrastructure.

Comparison with Previous Studies

Studies by Shetty et al. (2017) and Openja et al. (2022) have shown that Docker generates minimal overhead in executing GPU workloads, while offering a more controllable and reproducible environment. This study supports those findings by providing empirical evidence that, although Docker has higher GPU utilization, it still lags slightly behind native in terms of execution time.

Unlike most previous research, which focused on large-scale cloud environments, this study targets a local cloud context with a single GPU. Addressing a relevant gap for practitioners and researchers working in limited resources. Specifically, it provides insights for small-to medium-scale AI inference workloads.

DISCUSSION

Benchmarking results indicate that the native execution mode exhibits faster processing times compared to Docker, with average runtimes of 1.52 seconds and 2.55 seconds, respectively. This confirms that the overhead introduced by containerization systems such as Docker still impacts execution performance, even when GPU access has been optimized.

However, in terms of GPU utilization, Docker demonstrates higher and more consistent values, reaching an average of 86.2% and even peaking at 100% in several runs, while the native mode only averages around 45.6%. These findings suggest that Docker is able to allocate and utilize GPU resources more intensively for TensorFlow-based AI workloads.

This difference may be attributed to internal mechanisms in memory management and process execution adopted by TensorFlow within container environments. The high GPU utilization in Docker does not necessarily correlate with faster execution time due to the additional latency introduced by the virtualization layer.

From a practical standpoint, Docker is well-suited for deploying AI models that require consistent environments, scalability, and ease of distribution. In contrast, the native mode is preferable for local or experimental scenarios that demand maximum performance and faster execution times.

These results also align with previous findings, such as the study by Shetty et al. (2017), which reported that Docker introduces minimal overhead while offering advantages in consistency and dependency management. In the context of GPU consumer-grade local cloud environments, the contribution of this study lies in its empirical measurement of GPU performance and utilization, which has been rarely explored in prior research.

CONCLUSION

E-ISSN : 2807-9035
Volume 5, **Number** 2, November 2025
https://doi.org/10.47709/brilliance.v5i2.6794

This study compared GPU execution performance and utilization between native and Docker mode. A large matrix multiplication is used as the main workload using TensorFlow. The experiment resulted in native mode giving faster execution performance with an average of 1.52 seconds. In another hand, Dockers needs 2.55 seconds. Nonetheless, the Docker environment has better GPU utilization with an average 86,2%, while native can only reach 45.6%.

These findings highlight the trade-off introduced by containerization: while Docker incurs some execution overhead, it enables higher and more stable GPU utilization. Conversely, native execution proves to be more time-efficient, utilizing fewer GPU resources to complete the same task. Therefore, the decision to adopt Docker or native execution should align with specific use-case priorities—whether emphasizing minimal execution time or prioritizing environment portability and consistency.

REFERENCES

Belkhiri, A., & Dagenais, M. (2024). Analyzing GPU Performance in Virtualized Environments: A Case Study. *Future Internet*, *16*(3). https://doi.org/10.3390/fi16030072

Chang, C. H., Yang, C. T., Lee, J. Y., Lai, C. L., & Kuo, C. C. (2020). On construction and performance evaluation of a virtual desktop infrastructure with GPU accelerated. *IEEE Access*, *8*, 170162–170173. https://doi.org/10.1109/ACCESS.2020.3023924

Choi, H. S., Kim, Y., Lee, J., & Kim, Y. (2021). Empirical performance evaluation of communication libraries for MULTI-GPU based distributed deep learning in a container environment. *KSII Transactions on Internet and Information Systems*, *15*(3), 911–931. https://doi.org/10.3837/tiis.2021.03.006

Čisar, P., Erlenvajn, D., & Maravić Čisar, S. (2018). Implementation of software-defined networks using open-source environment. In *Tehnicki Vjesnik* (Vol. 25, pp. 222–230). Strojarski Facultet. https://doi.org/10.17559/TV-20160928094756

Kumar, M., & Kaur, G. (2022). Study of container-based JupyterLab and AI Framework on HPC with GPU usage. *2022 International Conference on Smart Generation Computing, Communication and Networking, SMART GENCON 2022*. https://doi.org/10.1109/SMARTGENCON56628.2022.10084107

Kurkure, U., Sivaraman, H., & Vu, L. (2017). Machine Learning Using Virtualized GPUs in Cloud Environments. In J. M. Kunkel, R. Yokota, M. Taufer, & J. Shalf (Eds.), *High Performance Computing* (pp. 591–604). Springer International Publishing.

Lingayat, A., Badre, R. R., & Gupta, A. K. (2018). Integration of linux containers in openstack: An introspection. *Indonesian Journal of Electrical Engineering and Computer Science*, *12*(3), 1094–1105. https://doi.org/10.11591/ijeecs.v12.i3.pp1094-1105

NVIDIA. (n.d.-a). *Nvidia Container Toolkit - Architecture Overview*. Retrieved August 4, 2025, from https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/arch-overview.html

NVIDIA. (n.d.-b). *Nvidia SMI*. Retrieved August 5, 2025, from https://developer.nvidia.com/system-management-interface

Oh, J., Kim, S., & Kim, Y. (2019). Toward an adaptive fair GPU sharing scheme in container-based clusters. *Proceedings - 2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems, FAS*W 2018*, 79–85. https://doi.org/10.1109/FAS-W.2018.00029

Openja, M., Majidi, F., Khomh, F., Chembakottu, B., & Li, H. (2022). Studying the Practices of Deploying Machine Learning Projects on Docker. *ACM International Conference Proceeding Series*, 190–200. https://doi.org/10.1145/3530019.3530039

Shea, R., & Liu, J. (n.d.). *On GPU Pass-Through Performance for Cloud Gaming: Experiments and Analysis*. https://doi.org/https://doi.org/10.1109/NetGames.2013.6820614

Shetty, J., Upadhaya, S., Rajarajeshwari, H. S., Shobha, G., & Chandra, J. (2017). An empirical performance evaluation of docker container, openstack virtual machine and bare metal server. *Indonesian Journal of Electrical Engineering and Computer Science*, *7*(1), 205–213. https://doi.org/10.11591/ijeecs.v7.i1.pp205-213

Shi, S., Wang, Q., Xu, P., & Chu, X. (2017). *Benchmarking State-of-the-Art Deep Learning Software Tools*. http://arxiv.org/abs/1608.07249

Tiying, F., & Zhengwei, L. (2019). *A GPU resource scheduling method and apparatus based on AI cloud*.

Walters, J. P., Younge, A. J., Kang, D.-I., Yao, K.-T., Kang, M., Crago, S. P., & Fox, G. C. (n.d.). *GPU Passthrough Performance: A Comparison of KVM, Xen, VMWare ESXi, and LXC for CUDA and OpenCL Applications*.

Wang, Y. E., Wei, G.-Y., & Brooks, D. (2019). *Benchmarking TPU, GPU, and CPU Platforms for Deep Learning*. http://arxiv.org/abs/1907.10701

Zhao, L., Jin, Y., Hu, G., Zhou, W., Wei, H., Li, R., Zhu, X., Xu, Y., Jin, J., & Li, Q. (2025). Design and Implementation of GPU Pass-Through System Based on OpenStack. *Computation*, *13*(2). https://doi.org/10.3390/computation13020038