Performance Characterization of Containerized DNN Training and Inference on Edge Accelerators

Anonymous Author(s)

***Abstract*—Edge devices have typically been used for DNN inferencing. The increase in the compute power of accelerated edges is leading to their use in DNN training also. As privacy becomes a concern on multi-tenant edge devices, Docker containers provide a lightweight virtualization mechanism to sandbox models. But their overheads for edge devices are not yet explored. In this work, we study the impact of containerized DNN inference and training workloads on an NVIDIA AGX Orin edge device and contrast it against bare-metal execution on running time, CPU, GPU and memory utilization, and energy consumption. Our analysis provides several interesting insights on these overheads.**

I. Introduction

Edge devices such as the Raspberry Pi, Intel Movidius and Google Coral have typically been used to perform lightweight vision-based DNN inferencing tasks in autonomous vehicles and smart city deployments. However, training on the edge is growing popular for two reasons. First is the advent of GPU-accelerated edge devices such NVIDIA Jetsons that approach GPU workstations in compute power [1]. E.g., the recent AGX Orin features a 12-core ARM CPU, an Ampere GPU with 2048 CUDA cores and 64 tensor cores, and $32GB$ of shared RAM. This delivers 275 TOPS of compute that matches an RTX 3080 Ti desktop GPU, while being smaller than a paperback novel and operating within $60W$ of power. Second, is the growing attention to data privacy and the rise of federated learning that trains DNN models on local data present on the edge, with only model weights sent to the server [2].

As the capabilities of edge devices increase, they are deployed as infrastructure compute resources, e.g., in smart cities, with the provision for multi-tenancy. This requires them to sandbox the DNN application, be it training or inferencing, from the host for privacy and security, and eventually from other concurrent applications [3]. Virtualization and containerization are two common techniques for such application packaging and sandboxing. Among these, Docker containers [4] offer a lightweight means to balance isolation and efficiency. While Docker is common in servers and workstations and its performance impact studied in detail, there are few investigations on the edge [5], [6], especially for training.

The unique features of Jetson edge devices, such as ARM-based CPUs, shared RAM between CPU and GPU, several power modes that control CPU, GPU and memory frequencies, and the lack of support for GPU partitioning across containers make them distinct from GPU servers and workstations. This, coupled with their more modest relative performance, makes it necessary to thoroughly examine the overheads of containerized DNN training and inference on accelerated edges.

In this paper, we make the following specific contributions:

1) We conduct detailed experiments on the NVIDIA Jetson Orin AGX devices for 3 representative DNN model–dataset combinations for training and inference using Docker containers and bare-metal.
2) We characterize the overheads of containerization on the running time, resource usage and energy consumption for DNN training and inference.
3) We investigate and isolate the sources of the overheads for the DNN models with different computational characteristics by studying various power modes on the device.

Our insights can help users make informed design choices to configure edge devices and select appropriate workloads.

II. Background

A. DNN training and inference pipeline

A generic *DNN training pipeline* has 3 stages – fetch, pre-process and compute. During the fetch phase, a mini-batch of data is fetched from disk to main memory. Then pre-processing happens on the CPU, where operations such as transformation or crop are done. Finally, the forward pass, backward pass and parameter update computation are done on the GPU. Training is run over all minibatches in an epoch, and several epochs are executed till convergence. In a typical *inference pipeline*, data may arrive continuously over the network and be grouped into a fixed-size batch in-memory, before it is pre-processed on the CPU and the inference computation, i.e., the forward pass, happens on GPU.

B. Docker containers

Docker [4] is a popular containerization platform that provides a convenient method to package and deploy an application and its dependencies. Docker containers share the device's operating system kernel, which makes them more lightweight than virtual machines. Containers use Linux *namespaces* to provide isolation between containers, and Linux *cgroups* to partition and limit the access of containers to system resources. Containers are created from a Docker image, which is generated by executing initialization commands in a Dockerfile. Docker containers do not need hardware virtualization support.

III. Related Work

A. Virtualization studies on edge devices

A few papers have examined virtualization on diverse edge devices. Roberto [7] study the performance of Docker on different Raspberry Pi and Odroid edge devices using various CPU, network, memory and disk benchmarks. Similarly, Hadidi et al. [5] evaluate the performance of containerized DNN inferencing on several low-end edge devices like the Raspberry Pi. RPi-class devices are much more constrained than Jetson devices which have GPU accelerators, faster CPUs and larger memory. Some studies [6] compare the performance of KVM and Docker on a Jetson TX2 using compute and network benchmarks. Divide and Save [8] presents a method to speed up computations by splitting up DNN inference workloads among multiple containers on the Jetson TX2 and AGX Orin. However, they only study CPU based inference and omit the GPU accelerator, which offers much of the compute power. None of these consider DNN training on accelerated edges.

B. Virtualization studies on servers and cloud

Xavier et al. [9] characterize the performance of various container and hypervisor-based virtualization techniques for HPC using compute, memory, network and disk benchmarks. Zhang et al. [10] characterize and predict the performance interference of GPU virtualization in cloud GPUs. It is interesting to note that GPU servers support various multi-tenancy and GPU partitioning mechanisms such as CUDA MPS (Multi Process Service) and MIG (Multi-Instance GPU). However, the Jetson class of edge devices does not support any of these mechanisms and time-shares the GPU among containers.

To the best of our knowledge, we are the first to perform a characterization of containerized DNN inference and training workloads on accelerated edge devices.

IV. Experiment Methodology

A. Hardware platform

We perform all our experiments on the Jetson AGX Orin developer kit [11], NVIDIA's latest and most powerful accelerated edge device. The AGX Orin has 12 ARM A78AE CPU cores, an Ampere GPU with 2048 CUDA cores and 32 GB of LPDDR5 RAM shared between the CPU and GPU. Its peak power is $60W$ and costs around 2000 USD. It comes with the OS installed on eMMC (flash based storage) and supports a variety of other storage media such as Micro SD card, USB HDD and NVME SSD. The full specifications of the device can be found in Table I. The AGX Orin offers a choice of several thousand custom power modes ($\approx 18k$), and each power mode can be thought of as a tuple of CPU cores, CPU frequency, GPU frequency and memory frequency. Unless otherwise mentioned, we run our experiments in the MAXN power mode III, where all components are set to their maximum possible frequencies. Dynamic Voltage and Frequency Scaling (DVFS) is off, and the onboard fan is set to maximum speed to avoid any temperature throttling effects. We use a $250GB$ NVME Samsung SSD 980 with a sequential read speed of about $3.5GBps$ to store the datasets, and this is exposed to containers as a Docker volume.

Table I: Specifications of NVIDIA Jetson AGX Orin devkit

Feature	AGX Orin
CPU Architecture	ARM Cortex A78AE
CPU Cores†	12
CPU Frequency (MHz)†	2200
GPU Architecture	Ampere
CUDA/Tensor Cores	2048/64
GPU Frequency (MHz)†	1300
RAM (GB)	32, LPDDR5
Storage Interfaces	μSD, eMMC, NVMe, USB
Memory Frequency (MHz)†	3200
Peak Power (W)	60
Price (USD)	$1999
Form factor (mm)	$110 \times 110 \times 71.65$

† This is the maximum possible value across all power modes. Actual value depends on the power mode used

B. Software libraries

The AGX Orin runs JetPack version 5.1, which comes with Linux for Tegra (L4T) $r35.2.1$, CUDA $v11.4.315$ and cuDNN $v8.6.0.166$. We use PyTorch $v2.0$ as the training and inferencing framework with torchvision $v0.15.1$. We set the *num_workers* to 4 in the PyTorch Dataloader to enable pipelined and parallel data fetch and pre-processing. Docker version 20.10.21 is used for containerization. We use the NVIDIA L4T PyTorch container for Jetpack as our base container image with additional libraries such as *jtop* for profiling various metrics. The container image runs the same JetPack, CUDA and cuDNN versions as the bare-metal. We also ensure that other libraries have consistent versions across bare-metal and containers.

C. Models and datasets

We choose three popular computer vision DNN models which perform image classification tasks for our experiments. These models provide different computational intensities and architectures and are representative of edge workloads. LeNet [12] is a simple Convolutional Neural Network designed to recognize handwritten digits, and we use this with the MNIST dataset. MobileNet [13] is another lightweight model designed for mobile devices using hardware-aware Network Architecture Search techniques, and we use the MobileNetv3large variant with a subset of the Google Landmarks Dataset (GLD23k). We also use two Residual Neural Networks [14] - a smaller variant ResNet-18 for training and a larger variant ResNet-50 for inference, both with a subset of the ImageNet dataset. The full details of the models and dataset are listed in Table II.

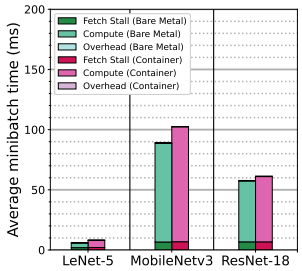
D. Performance metrics

We measure and report several system parameters such as CPU and GPU utilization, RAM usage and power (sampled every $1s$). We use the *jtop* Python module (a wrapper around the *tegrastats* utility from NVIDIA) to measure CPU and GPU utilization and power. CPU utilization is reported as the average across all 12 CPU cores, and GPU utilization as the average across the 2 Graphic Processing Clusters (GPCs).

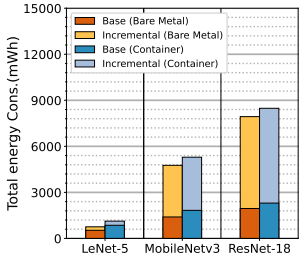
Table II: DNN Models and Datasets

Model	# Layers	# Params	FLOPS†	Dataset	# Samples	Size on Disk	Minibatch Size	Used for
LeNet-5	7 [12]	$60k$	$4.4M$	**MNIST**	$60,000$	$46MB$	16	Train, Inf
MobileNet v3	20 [13]	$5.48M$	$225.4M$	**GLD23k**	$23,080$	$2.82GB$	16	Train, Inf
ResNet-18*	18 [14]	$11.68M$	$1.82G$	**ImageNet**	$50,000$	$6.74GB$	16	Train
ResNet-50	50 [14]	$26M$	$4G$	**ImageNet**	$50,000$	$6.74GB$	16	Inf

† As per the typical practice, FLOPS reported corresponds to a forward pass with minibatch size 1.
* We use a smaller ResNet for training to avoid running out of memory

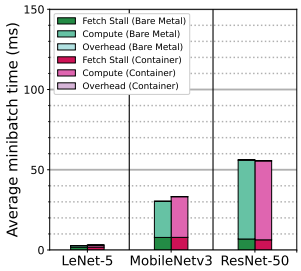


(a) Average minibatch time

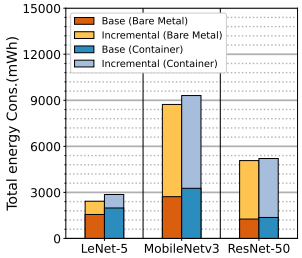


(b) Total energy

Figure 1: Avg. minibatch time & total energy for DNN training

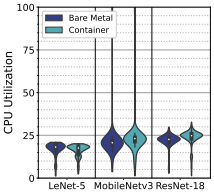


(a) Average minibatch time

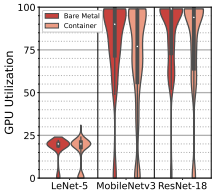


(b) Total energy

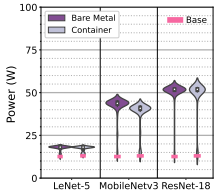
Figure 3: Avg. minibatch time & total energy for DNN inf.



(a) CPU utilization

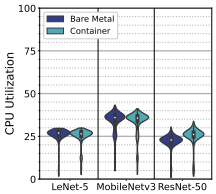


(b) GPU utilization

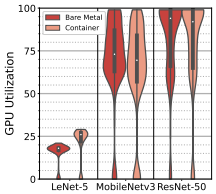


(c) Power

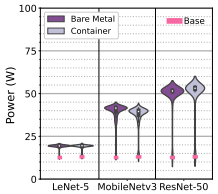
Figure 2: CPU, GPU utilization and power for DNN training



(a) CPU utilization



(b) GPU utilization



(c) Power

Figure 4: CPU, GPU utilization and power for DNN inference

Power measurements are based on the onboard sensors that capture the module load, and these readings are aggregated over the runtime of the workload to capture total energy. The Linux utility *free* is used to measure used, free and cached memory. We instrument the PyTorch code to measure fetch stall, GPU compute and end-to-end times for every training and inference minibatch. Fetch stall time is the time taken for fetching and pre-processing data that does not overlap with the GPU compute, and results in the GPU being idle. GPU compute time is measured using `torch.cuda.event` with `synchronize` to accurately capture the minibatch's execution time on the GPU. We also measure the end-to-end execution time of the minibatch and report our logging overhead as the difference between end-to-end time and the sum of fetch stall and GPU compute times. We have performed all experiments 2-3 times to ensure reproducibility.

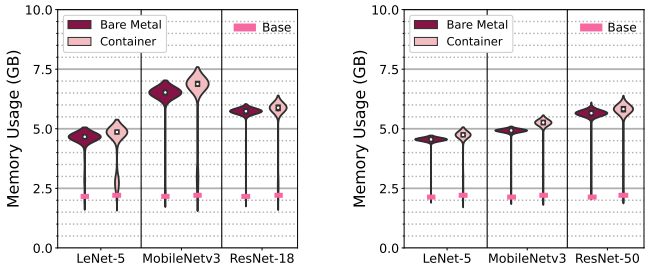
V. Results and Analysis

A. DNN training

We run each training model on bare-metal and container and report the average minibatch time, total energy, CPU & GPU utilization, power and memory usage. Every training workload is run for at least 3 epochs. LeNet runs very fast, therefore we run it for 5 epochs to ensure that it runs at least for a few minutes. ResNet and MobileNet run for 3 epochs. We use a minibatch size of 16 and SGD as the optimizer for all models following standard implementations. Our analysis is presented as a series of takeaways.

1) Lightweight training workloads see a significant increase in compute time when run in a container as opposed to bare-metal: In Fig 1a, we report average minibatch time as a stacked plot of fetch stall, compute time and logging overhead (minimal, not seen in plots). As seen from the plot, there is an increase in compute time from bare-metal to container for all 3 models. For LeNet, the compute time (2nd stack in the plot) increases from $3.91ms$ to $6.01ms$, an increase of 53.7%. MobileNet sees an increase from $82.12ms$ to $95.48ms$, which is 16.26%, and ResNet sees an increase from $50.53ms$ to $54.24ms$, which is 7.34%. In terms of FLOPS (Table II), LeNet is the most lightweight of the 3, followed by MobileNet and then ResNet. As can be seen, the increase in compute time follows the same pattern and is more for lightweight models LeNet and MobileNet.

2) This increase in compute time is due to CPU overheads of containerization: In order to localize the source of the compute increase, we look at violin plots of CPU, GPU utilization and power reported in Figs 2a, 2b and 2c respectively. Looking at MobileNet, we see that the median GPU utilization falls from 90% on the bare-metal to 77% on the container, indicating that the GPU is spending time waiting. The lower GPU utilization also causes the median power for MobileNet to fall slightly from $43.7W$ to $40.8W$ as the GPU is major contributor to power. We also notice a higher median CPU utilization of 23% on the container as opposed to 20.67% on bare-metal. This leads us to make the observation that the CPU is the source of the compute



(a) Training- memory footprint (b) Inference- memory footprint

Figure 5: Memory footprint for DNN training and inference

overhead. For ResNet, the CPU utilization increase and GPU utilization decrease follow the same trend but are much less pronounced, indicating lesser compute overhead. Since LeNet is a very small model with very low GPU utilization, we do not observe a difference here. Most of the power for LeNet is contributed to by the baseload of the system as seen in Fig 2c, and we don't see a difference here too.

3) This increase in compute time is further exacerbated when CPU frequency is lowered using a custom power mode, confirming that the CPU is the source of the overhead: To verify that the CPU is indeed the source of the overhead, we define 3 custom power modes as listed in Table III and run the training workloads for these 3 power modes apart from the default power mode (MAXN). Each of the power modes lowers one of CPU, GPU and memory frequencies while keeping the others the same as the default power mode. LeNet and MobileNet container runs show a high sensitivity to lowering of CPU frequency more than GPU or memory–i.e. the compute time increases more sharply for container training when CPU frequency is lowered. In going from power mode MAXN to PM1_CPU, the compute time for MobileNet on bare-metal increases from $82.12ms$ to $91ms$, an increase of 10.7%. In contrast, for the container, the increase in compute time is from $95.4ms$ to $130ms$, which is a much higher increase of 36.1%. We do not see this with other power modes. This proves conclusively that there is a CPU overhead of containerization which affects lightweight models more, and this effect gets worse if run in a power mode with reduced CPU frequency.

4) Why the overhead?: To further narrow down the reason for the CPU overhead, we traced the system calls for MobileNet training on bare-metal and container using the Linux *strace* utility. We observed a larger number of cumulative system calls for the container. Since this pointed to a system call related overhead, we ran the container with *seccomp* turned off to observe the effect of Docker's system call filtering. However, we did not observe any change in runtimes from the previous experiments. On a parallel track, we added more fine-grained profiling to the compute phase of the workload, recording times for forward pass, backward pass and the parameter update (`optimizer.step` in PyTorch). This breakup of the minibatch time is reported in Fig 7. We noticed that among the three compute steps, the parameter update using the SGD optimizer sees the most increase from $2.63ms$ to $17.63ms$, (a $6.7\times$ increase as compared to 11.4% increase for forward and 4.89% increase for backward), and we are further investigating this. We also notice that container runs take longer after adding this fine-grained instrumentation and this could be due to asynchronous operations being affected by the `torch.synchronize` needed for recording the times.

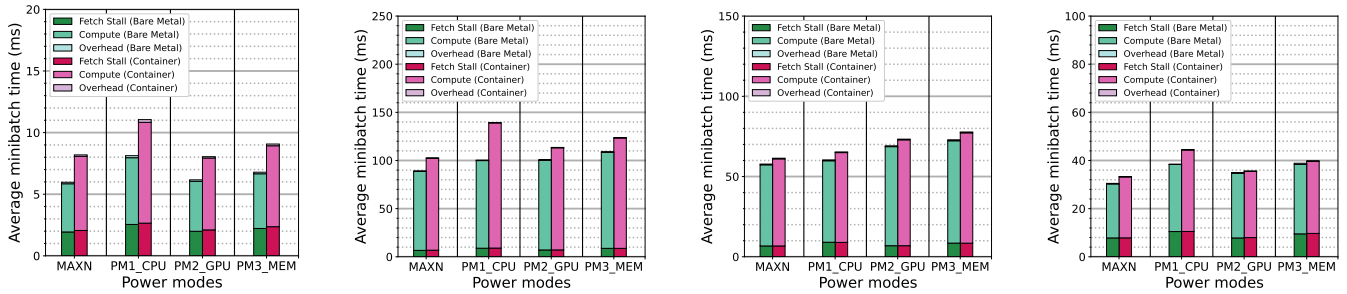
5) Energy consumption of the containerized training workload is slightly higher than on bare-metal, more so for lightweight models: In Fig 1b, we report the energy consumed by the 3 models for training as a stacked bar plot. The bottom stack of the bar represents the baseload, which is the energy it takes for the system to run with no workload. To measure this, we run only our logging script for $10min$ on bare-metal with no workload running. Similarly, we spin up a container and run the logging script with no workload for $10min$. We scale both these energy values by the runtime of the actual workload on the bare-metal and container respectively. The top stack reports the incremental energy spent for training i.e, energy over and above the baseload that is needed for model training. Again, we observe that there is an increase in total energy consumption for containerized training as compared to bare-metal. This increase is around 48.89% for LeNet, 11.11% for MobileNet, and 6.84% for ResNet, more significant for lightweight models.

6) This increase in energy for lightweight models is due to running for a longer time at a lower GPU utilization: As seen from Fig 1a, there is an increase in average minibatch time. The increase in energy is majorly due to the baseload increase of running for a longer time. For example, for MobileNet, we see a baseload increase of 30.76% along with an incremental energy increase of 2.94%. This indicates that running for longer at a lower GPU utilization is not beneficial i.e, the power benefits obtained by running at a lower GPU utilization are outweighed by the energy consumed by running longer.

7) The memory footprint of containerization is minimal for training workloads: In Fig 5a, we report the memory used on bare-metal v/s container as a violin plot. We also report the base memory usage of the system without any workload running using markers. As expected, we do not observe a large memory overhead for containers. The median memory increases from $4.66GB$ to $4.86GB$, an overhead of $0.2GB$ for LeNet. Similarly, MobileNet and ResNet see an increase of $0.35GB$ and $0.15GB$ respectively. This indicates that multiple containers can be spun up without incurring a significant memory overhead even on resource-constrained edge devices.

B. DNN inference

We run each inference model on bare-metal and container and report the average minibatch time, total energy, CPU & GPU utilization, power and memory usage. We simulate the effect of images arriving over the network by running a Dataloader iteration before the start of the workload and verify that all images are in memory. Every inference workload is run for at least a few minutes, and the number of minibatches is chosen to ensure this. ResNet is run for 6250 minibatches,



(a) Training - LeNet-5 (b) Training - MobileNet v3 (c) Training - ResNet-18 (d) Inference - MobileNet v3

Figure 6: Impact of custom power modes on average minibatch time of DNN training and inference

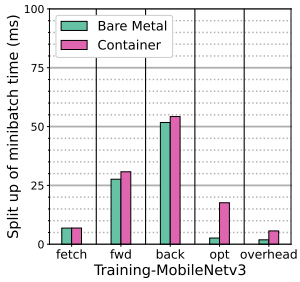


Figure 7: Compute time break-up for MobileNet training

MobileNet for 24600 minibatches, and LeNet for 131250 minibatches. Minibatch size is set to 16 for all.

1) Lightweight inference workloads also see an increase in compute time, but the increase is less significant as compared to training: In Fig 3a, we report average minibatch time as a stacked bar plot of fetch stall, compute and overhead times. We observe that compute time increases by 30.52% for LeNet and 12.93% for MobileNet. ResNet does not have a noticeable increase in compute time (less than 1%). This can be explained by the fact that inference involves only the forward pass, and the parameter update which caused most of the overhead in training is not a part of inference workloads. We also perform experiments with the custom power modes for MobileNet and report the average minibatch times in Fig 6d. For the power mode with lowered CPU frequency (PM1_CPU), we observe that the compute time for the container increases by 33.72% while for the bare-metal it increases by 24.26% as compared to MAXN. The compute increase for containerized inference is lesser compared to training.

2) Correspondingly, the energy increase is also modest for inference: In Fig 3b, we report the energy consumed by the 3 models for inference as a stacked bar plot of baseload and incremental energy. The increase in total energy is 18.1%, 6.63% and 2.68% for LeNet, MobileNet and ResNet respectively, which is much lower than in training. This is due to the lower overheads of compute time in inference. Again, most of the energy increase comes from the baseload. For instance, MobileNet sees an increase of 20.03% in the baseload and 0.5% in incremental energy.

3) The memory footprint of Docker for inference is also minimal: The memory footprint of containerized inference is higher by $0.19GB$, $0.34GB$ and $0.17GB$ for LeNet, MobileNet and ResNet respectively as compared to bare-metal.

Table III: Power Modes Evaluated on AGX Orin

Label	CPU Cores	CPU MHz	GPU MHz	RAM MHz
MAXN	12	2201.6	1301	3200
PM1_CPU	12	**1497.6**	1301	3200
PM2_GPU	12	2201.6	**930.75**	3200
PM3_MEM	12	2201.6	1301	**2133**

Cells in bold indicate a value change from the cell in the first row.

VI. Conclusions and Future Work

In this paper, we closely examine and characterize containerized DNN training and inference workloads. We demonstrate that lightweight DNN models incur overheads in compute time and consequently energy when containerized. These overheads show up even more in power modes with lower CPU frequencies. This is relevant to federated learning workloads, which often involve very lightweight models. In future, we plan to investigate the effect of containerization when inference and training are run concurrently, as this is an emerging usecase of continuous or lifelong learning.

References

[1] Prashanthi S.K, S. A. Kesanapalli, and Y. Simmhan, "Characterizing the performance of accelerated jetson edge devices for training deep learning models," *POMACS*, vol. 6, no. 3, December 2022.

[2] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, 2017.

[3] K. Shmelkov, C. Schmid, and K. Alahari, "Incremental learning of object detectors without catastrophic forgetting," in *IEEE ICCV*, 2017.

[4] Docker, "Docker," https://www.docker.com/, 2022.

[5] R. Hadidi *et al.*, "Characterizing the deployment of deep neural networks on commercial edge devices," in *IEEE IISWC*, 2019.

[6] B. Wang *et al.*, "Enabling high-performance onboard computing with virtualization for unmanned aerial systems," in *ICUAS*, 2018.

[7] R. Morabito, "Virtualization on internet of things edge devices with container technologies: A performance evaluation," *IEEE Access*, vol. 5, 2017.

[8] A. Khoshsirat, G. Perin, and M. Rossi, "Divide and save: Splitting workload among containers in an edge device to save energy and time," *arXiv preprint arXiv:2302.06478*, 2023.

[9] M. G. Xavier *et al.*, "Performance evaluation of container-based virtualization for high performance computing environments," in *PDP*, 2013.

[10] X. Xu, N. Zhang, M. Cui, M. He, and R. Surana, "Characterization and prediction of performance interference on mediated passthrough GPUs for interference-aware scheduler," in *USENIX HotCloud*, 2019.

[11] Nvidia., "Jetson agx orin developer kit," https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/, 2022.

[12] Y. Lecun *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, 1998.

[13] A. Howard *et al.*, "Searching for mobilenetv3," in *IEEE ICCV*, 2019.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE CVPR*, 2016, pp. 770–778.