# Measuring the Energy Consumption of Docker Images for ML Workloads

**Ana Țerna, Andrei-Iulian Vișoiu, Lucian Toșa, Monica Păun**
Delft University of Technology

## Abstract

The growing adoption of Artificial Intelligence across all industries has led to concerns over its carbon footprint. Energy consumption is particularly important during the inference phase, which is often run continuously and at scale. Docker containerization is the industry standard for deploying ML workloads due to its flexibility and portability, but the specific impact of image configurations on energy efficiency has not been explored yet. This study addresses this gap and investigates how different image configurations affect consumption during inference. We use a ResNet-50 image classification model to perform inference on randomly generated images of various resolutions, while measuring energy consumption with EnergiBridge under controlled conditions. Five Docker setups with increasing base image complexity ranging from base OS images to optimized NVIDIA containers were evaluated. We have found an indication that integrated images perform better than manually installed ones. However, some pre-built images are substantially larger than others, and this is also a factor to take into account. For future research directions, we recommend further benchmarks on varying hardware to generalise the results, and exploration into which Docker layers affect consumption the most.

## 1 Introduction

Artificial Intelligence (AI) has rapidly transformed industries by enabling automotive decision-making, advanced pattern recognition and predictive analytics. However, the increased adoption of AI comes with a significant concern: its energy footprint. Training large-scale models requires substantial computational power, often running for days or weeks on high-performance hardware, consuming vast amounts of electricity. As a result, these models are costly to train and develop both financially but also environmentally. For instance, training deep-learning models can produce as much carbon footprint as flights do while traversing continents [1]. While the training part of AI models has received much attention in energy consumption research [2], inference, the stage where models are deployed and used to generate real-time predictions, also contributes to this footprint, particularly when scaled across multiple devices [3].

Machine Learning (ML) inference workloads are widely used in various applications from ranking systems to autonomous driving, where models generate predictions based on the patterns and knowledge gained during the training phase. Unlike training, inference is often performed continuously, leading to a cumulative energy demand that can exceed the cost of training. Therefore, it is essential to explore strategies for minimizing both the cost and energy consumption of this phase. This study focuses on a specific aspect of this challenge: understanding the impact of containerization on the energy efficiency of ML inference workloads.

Containerization is a widely adopted approach for deploying ML workloads due to its portability, scalability of resources and ease of configuration [4]. Specifically, containers package ML models, along with the necessary dependencies and hardware configurations, which consequently, allow developers to achieve consistency in resource management across different computing environments. However, while containerization offers significant benefits for deployment efficiency, it also introduces additional computational overhead due to factors such as resource isolation and container runtime processes that occur while a workload is running. These factors can impact execution time, CPU usage, and memory consumption, all of which influence the overall energy efficiency of an ML workload.

Among the various containerization technologies available, Docker has emerged as one of the most widely used tools for deploying ML workloads. This popularity is due to its ability to streamline the deployment process, enabling the creation of lightweight, portable containers that can run across diverse computing environments. However, while Docker facilitates the management of dependencies and offers scalability, the specific impact of Docker image configurations on the energy efficiency of ML workloads remains underexplored. With these aspects in mind, we raise the following research questions:

- **RQ1**: How do different Docker images affect the energy consumption of inference ML workloads?
- **RQ2**: What are the configurations in Docker images that affect energy consumption the most?

This study aims to empirically measure the energy consumption of different Docker images when running an ML inference workload. For that, we will conduct an analysis of the execution time and total energy consumption of several Docker images specifically used for ML workloads. Through this analysis, we seek to determine whether the choice of Docker image influences energy consumption and, if so, which configurations contribute to higher resource costs in AI deployments. Lastly, this study aims to provide guidance for developers and research in improving the deployment aspect of ML workloads while also addressing the growing concern of AI's environmental impact.

The paper is structured as follows. Section 2 reviews prior work investigating the energy efficiency of deployment frameworks for ML workloads. The chosen methodology will be presented and justified in Section 3. Results will be presented in Section 4, with further discussions in 5. Section 6 will present the final conclusions of the study.

## 2 Background & Related Work

This section presents a brief overview of prior research that investigates the performance and energy efficiency of various deployment tools and frameworks for ML workloads.

Docker containers[1] have emerged as a popular solution for managing and executing ML workloads due to their ability to package all required dependencies into a portable and reproducible environment. A Docker container runs based on an image defined in a Dockerfile, which contains specifications such as the base operating system, libraries, and application code. The efficiency of a Docker image depends on the choices made during the configuration phase and certain decisions can lead to larger, more complex images that consume additional energy at runtime [5].

Although containerization introduces lower overhead compared to full virtualization, it still imposes an energy cost compared to bare-metal deployment. Studies have explored how Docker-based workloads consume energy under different conditions. Warade et al. [6] measured the energy footprint of various Docker containers running common web and database workloads. The choice of a base image for a Docker container was also shown to have a significant impact on the energy consumption when tested on Redis workloads [7]. Their findings highlight the significant energy consumption of Dockerized applications, particularly under high load conditions, and emphasize the need for energy-aware optimization strategies.

Based on the aforementioned studies, a non-trivial aspect of building efficient Docker images is the selection of the base image. As shown in Figure 1, the starting point for building a Docker image is the choice of the base image. The base image is usually imported from DockerHub[2] (e.g., ubuntu, node, python) and provides an operating system (OS) or preinstalled environment applications. The choice of the base image significantly impacts the energy consumption in several ways. First, larger base images contribute to longer startup times, consuming more energy during initialization. Some base images also include background services or dependencies, which further increases the energy consumption. A recent study [8] analyzed various OS base images (i.e., Debian, Ubuntu, CentOS, and Alpine) on several workloads (e.g., web-based, ML, DB-based and gaming workloads) and found out that there is a significant difference in energy consumption between base images for most of the workloads. For instance, some base images perform efficiently for one workload but consume significantly more time and energy for other types of workloads. This shows that the optimal choice of the base image depends on the containerized workload, however, there has been little research done in analyzing how various base images impact the energy consumption specifically for ML workloads.

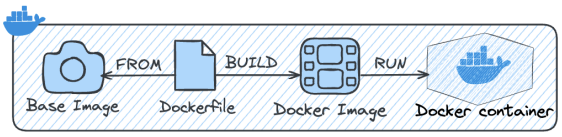


Figure 1: The components that are used for setting up and running a Docker container where each component can contribute to the energy consumption of dockerized workloads.

Prior research has examined the impact of containerization on ML workloads from different angles. Alizadeh and Castor [9] investigated whether the choice of an ML framework influences inference energy consumption but found no strong correlation. Park et al. [10] analyzed how resource contention in multi-tenant environments affects deep learning performance and efficiency, identifying potential degradation due to container resource conflicts. Hampau et al. [11] compared multiple deep learning deployment strategies and found that dockerized workloads consumed more energy than other approaches due to the overhead introduced by the containerization layer.

Containerization remains the industry standard for deploying ML models due to its flexibility and portability, however, it is important to first understand its energy impact in detail. Currently, there remains a significant research gap in understanding the exact factors influencing the energy consumption of containerized workloads and how to optimize these environments efficiently. Warade et al. [6] further underscore this concern by demonstrating that even common Docker containers exhibit non-trivial energy consumption patterns. Their work provides a foundation for future research aimed at optimizing Docker deployments for energy efficiency, particularly in ML workloads where compute-intensive tasks exacerbate power consumption.

While these studies contribute valuable insights into the energy consumption of workloads under different deployment environments, a notable gap remains in understanding how specific Docker configurations influence energy efficiency. Few studies have directly addressed the relationship between Docker image configurations and the energy consumption of ML inference workloads, which is important for understanding the main picture of the energy footprint of containerized AI deployments.

[1] http://docker.com/

[2] https://hub.docker.com/

## 3 Methodology

In this section, we outline the method taken to address the questions exposed in Section 1. The approach relies on constructing multiple docker images for an ML workload, which are later subjected to energy measurement.

To measure the energy consumption of a machine learning workload, we used a standard deep learning pre-trained model for image classification, ResNet-50, part of the Residual Networks family introduced by He et al. [12]. The following experiment was designed based on the study of deep learning benchmarks as documented in the AI Power Meter [3].

The first step of the benchmark is to choose between running the experiment on either the GPU or CPU. However, for the purposes of this experiment, the aim was to run the benchmark on an NVIDIA GPU. For the study of energy consumption in inference, we used a pre-trained ResNet-50 model from Pytorch [4]. We then attempt to perform inference with this model on images with resolutions from 256x256 up to 2048x2048. For each image resolution, we generate a purely random 3-color image, which we then normalize according to PyTorch's guidance for the ResNet50 model. While we could have used a pre-existing dataset, like ImageNet [5], we have opted for generating random colored images instead. This option is also used by AI Power Meter, and it can help determine the worst-case energy consumption. This is because real images contain structured patterns that can activate the deep neural network in predictable ways. While this is not necessarily indicative of real-world use case scenarios, the aim of this study is to determine the energy consumption of different Docker images using the same workload, therefore, a worst-case scenario is more suitable.

After normalizing, we perform inference for a set amount of times on this image:

- 400 times for the image with resolution 256x256.
- 200 times for the image with resolution 512x512.
- 133 times for the item with resolution 1024x1024.
- 100 times for the item with resolution 2048x2048.

This amount of iterations for each image has been chosen for computational reasons, as the GPU time increases exponentially with the resolution of the image.

The experiment was run with five different setups for the Docker images. The main distinctions between these were the base image and the package installation methods. The five images are built up based on increasing complexity, having the following configurations:

- Ubuntu: For the base image we started with Ubuntu 22.04 upon which Python 3.10, CUDA 12.6 and PyTorch 2.6 were installed. Uncompressed size is 9.11GB.
- Python: For this Docker image, we have used the base image to be Python 3.10, installing after that CUDA 12.6 and PyTorch 2.6 manually. Uncompressed size is 9.95GB.
- NVIDIA CUDA: This builds on top of the nvidia/cuda base image which comes with the CUDA toolkit pre-installed. Then we installed Python 3.10 and PyTorch 2.6. It has a size of 12.91GB.
- PyTorch CUDA: The base image is pytorch/pytorch which comes with all our required packages installed. Resulting size is 9.62GB
- NGC PyTorch (nvcr-base): This is an optimized image from NVIDIA's NGC catalog with all packages already installed. This is an enterprise-grade image that NVIDIA recommends for production workloads. It is also the largest image, at 40.86GB.

These five Docker images and their corresponding docker files were designed in this way as we wanted to explore the impact of having to manually install software and packages on top of a bare base image and compare this to gradually simpler images where the required components come preinstalled and configured with the base image.

Firstly, the image built with the Ubuntu base image was chosen, as it starts off from a very basic image with none of the components we needed installed and serves as the starting point of our experiments. This gave us complete control over the installation method of the required packages and libraries, which can help identify if packages installed manually lead to better efficiency as opposed to packages coming pre-installed in base images. While we could have opted for even simpler images like Alpine Linux or RHEL UBI, Ubuntu is at the foundation of the other 4 base images we have used. Therefore, this choice eliminates some variability from our experiments.

The next four images are designed so that they come with gradually more components pre-installed in the base image, reducing the amount of manual installations, which will help us identify which component brings the most efficiency gains depending on its installation method. Therefore, the image based on the Python base image aims to check if a manual installation of Python is more efficient or not than a prebuilt base image. The same approach is followed with the image based on NVIDIA CUDA, which will help us verify whether a manual installation of CUDA is more inefficient.

Lastly, the image based on PyTorch CUDA is the most comprehensive and comes with all the components pre-installed. This eliminates any manual installation, and is therefore a good comparison point with the first image in our experiment, which only contains manual installations. As an extra addition, we have tested the more optimized version from the NVIDIA NGC catalog, which also comes pre-installed with all the components, but is advertised as an enterprise-grade image. This can help us compare the two and identify whether there are certain optimizations that NVIDIA developed in their base images.

To measure the energy consumption of Docker images, we utilized the EnergyBridge [6] tool for power measurement and applied the experiment framework depicted in Figure 2 to ensure consistent and controlled execution conditions. Each experiment began with a one-minute CPU-intensive warm-up

[3] https://greenai-uppa.github.io/AIPowerMeter/index.html

[4] https://pytorch.org/hub/nvidia_deeplearningexamples_resnet50/

[5] https://www.image-net.org/

[6] https://github.com/tdurieux/EnergiBridge

phase, designed to stabilize processor behavior and minimize variability in energy measurements. Following the warm-up, the actual dockerized workload was executed within a subprocess, allowing precise monitoring of energy usage without interference from external system processes. To ensure statistical reliability, each Docker image was tested across 30 independent runs, with one-minute breaks between executions to allow the system to return to a baseline state. Moreover, we addressed various variables such as stable room temperature and internet connection and throughout the experiment, the computer was always plugged into a power source with its brightness lowered.

For reproductibility purposes, the setup can be found in our GitHub repository[7].

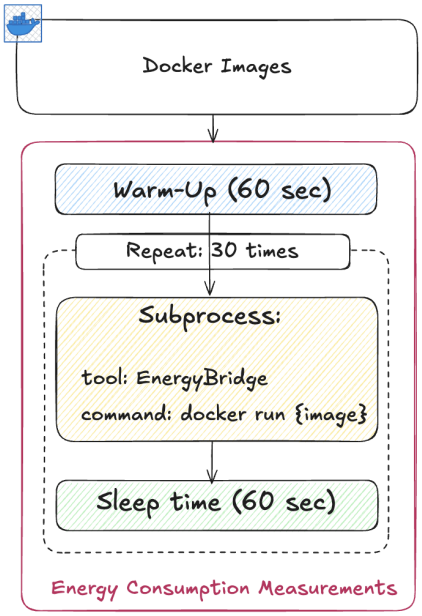


Figure 2: Energy measurement framework utilized for running and recording the metrics related to the energy consumption of various Docker images.

### 3.1 Harware Setup

All experiments were conducted on a machine running the Linux operating system (Ubuntu 24.04.2 LTS). The hardware configuration included an Intel Core i5-6500 quad-core processor, 16 GiB of RAM, and an NVIDIA GeForce GTX 970 GPU. Each experiment was executed via a Python script, which invoked the necessary commands using the *subprocess*[8] module to ensure isolation and a controlled execution environment.

## 4 Results

In this analysis, we first performed the Shapiro-Wilk normality test to assess whether the energy consumption data for each variant followed a normal distribution. Upon examination, we found that not all variants were normally distributed. Consequently, we opted to conduct the Kruskal-Wallis test to determine if there were any significant differences between the groups of non-normally distributed data. The p-value from this test was close to zero, indicating a significant difference across the groups. However, this test only indicated that differences exist but did not specify which groups differed from each other. To identify the specific pairs of groups with significant differences, we applied Dunn's test as a post-hoc analysis. We chose Dunn's test over Tukey's HSD because of the non-normal distribution of some variants, making it a more appropriate choice for non-parametric data. After confirming that there were significant differences between the groups, we used Cliff's Delta to measure the magnitude of these differences. Additionally, we computed standard descriptive statistics, including the mean, median, and standard deviation, which provide useful insights into the central tendency and spread of the data. While the mean offers an overall average, the median gives a better representation of the central value for skewed distributions, helping us understand the data better.

The energy consumption results are shown in Table 1 and Figure 3, with execution times presented in Table 3 and Figure 4. We can notice that *pytorch-base* had the lowest energy consumption with moderate variance, followed by the *cuda-base* image, which also contains an outlier in its data. We also can see that the execution time plot in Figure 4 of the ResNet-50 workload has the same pattern as the one of the energy consumption.

| Variant | Median | Mean | $\sigma$ |
|---|---|---|---|
| pytorch-base | 37347.75 | 37432.56 | 386.70 |
| cuda-base | 37802.63 | 37914.54 | 731.44 |
| nvcr-base | 38351.47 | 38327.34 | 417.68 |
| ubuntu-base | 38432.59 | 38481.38 | 353.91 |
| python-base | 38478.95 | 38510.93 | 323.73 |

Table 1: Energy consumption statistics for different base variants obtained by running the ML workload, sorted by median.

The energy consumption statistics from Table 1 show that, while the central tendencies of the variants are somewhat similar, there is noticeable variation in the *cuda-base* image, which does suggest the presence of outliers in its energy consumption data. To further understand if the differences among the energy consumptions are significant, we perform the Shapiro-Wilk normality test shown in Table 2. Based on this, we notice that *cuda-base* and *pytorch-base* did not follow a normal distribution, as indicated by their p-values (i.e., below 0.05), whereas *python-base*, *nvcr-base*, and *ubuntu-base* exhibited normality.

To assess the significant differences in energy consumption among the variants and also to answer *RQ1: "How do different Docker images affect the energy consumption of inference ML workloads?"*, Dunn's post-hoc test with Bonferroni correction was applied with results present in Table 4. The results indicate that there are significant differences in energy

[7]https://github.com/anaterna/sustainablesoftware/tree/main/project2

[8]https://docs.python.org/3/library/subprocess.html

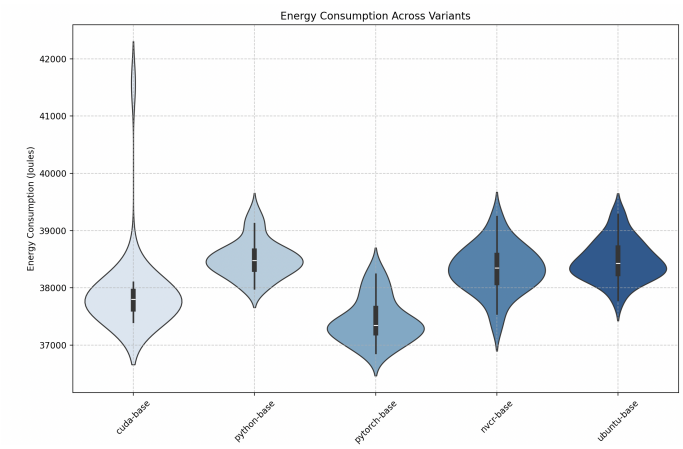


Figure 3: Energy consumption during the workload.

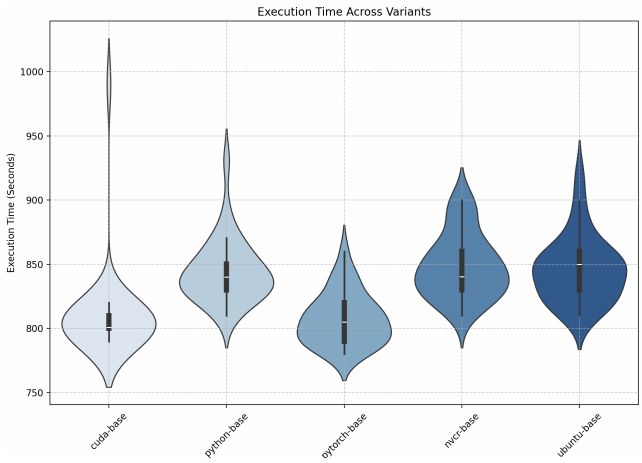


Figure 4: Execution time during the workload.

| Variant | p-value | Normality |
|---|---|---|
| cuda-base | 0.0 | ✗ |
| nvcr-base | 0.98815 | ✓ |
| python-base | 0.14876 | ✓ |
| pytorch-base | 0.04535 | ✗ |
| ubuntu-base | 0.38089 | ✓ |

Table 2: Shapiro-Wilk normality test results. A ✓ indicates normal distribution, while a ✗ indicates non-normal data.

consumption between several of the base variants. Specifically, *cuda-base* and *pytorch-base* show lower energy consumption compared to *ubuntu-base* and *python-base*. The Dunn's test, which was applied due to the non-normality of some of the data (as determined by the Shapiro-Wilk normality test), is suitable for comparing multiple groups when the assumption of normality is not met. The differences observed in the energy consumption across these base images may be attributed to the underlying configurations and optimizations in each image. For instance, *cuda-base* and *pytorch-base* likely have optimizations that lead to lower energy usage, especially for machine learning workloads, whereas *ubuntu-base* and *python-base* might not have the same level of such optimizations, leading to higher energy consumption. These variations highlight the importance of choosing the right base image depending on the specific workload and resource efficiency requirements.

| Variant | Median | Mean | $\sigma$ |
|---|---|---|---|
| cuda-base | 800.92 | 809.79 | 35.27 |
| pytorch-base | 805.06 | 807.24 | 20.24 |
| nvcr-base | 840.27 | 848.78 | 24.77 |
| python-base | 840.07 | 844.48 | 24.69 |
| ubuntu-base | 850.05 | 846.00 | 25.87 |

Table 3: Execution time statistics for different base variants, ordered by median.

Table 4: Dunn's Test Results (Bonferroni-corrected p-values)

| | nvcr | python | pytorch | ubuntu |
|---|---|---|---|---|
| **cuda** | **0.0011** | **0.0000** | 0.3343 | **0.0000** |
| **nvcr** | - | 1.0000 | **0.0000** | 1.0000 |
| **python** | - | - | **0.0000** | 1.0000 |
| **pytorch** | - | - | - | **0.0000** |

Table 5: Effect Size Analysis: Mean Difference and Cliff's Delta

| Variant 1 | Variant 2 | Mean Difference | $\Delta$ |
|---|---|---|---|
| cuda-base | python-base | -596.39 | **-0.86** |
| cuda-base | pytorch-base | 481.98 | **0.60** |
| cuda-base | nvcr-base | -412.80 | **-0.69** |
| cuda-base | ubuntu-base | -566.84 | **-0.84** |
| python-base | pytorch-base | 1078.37 | **0.97** |
| python-base | nvcr-base | 183.59 | 0.25 |
| python-base | ubuntu-base | 29.55 | 0.08 |
| pytorch-base | nvcr-base | -894.78 | **-0.87** |
| pytorch-base | ubuntu-base | -1048.82 | **-0.94** |
| nvcr-base | ubuntu-base | -154.04 | -0.22 |

To further understand the differences between images, we look at the mean difference and Cliff's delta ($\Delta$), which offers deeper insights between various base image pairs. Notably, large negative and positive values of Cliff's delta suggest substantial differences between some image pairs. For example, the comparison between *cuda-base* and *python-base* shows a large negative mean difference of -596.39 J, accompanied by a significant negative value of (i.e., $\Delta = -0.86$), indicating a considerable reduction in energy consumption in *cuda-base*. Similarly, the *cuda-base* and *ubuntu-base* comparison also demonstrates a significant effect size (i.e., $\Delta = -0.84$), further emphasizing the potential energy efficiency of *cuda-base*. Conversely, *python-base* and *pytorch-base* exhibit a large positive mean difference (i.e., 1078.37 J) and an even higher positive delta (i.e., $\Delta = 0.97$), highlighting a no-

table increase in energy consumption with *python-base* compared to *pytorch-base*. The magnitude of differences between *pytorch-base* and *ubuntu-base* (i.e., $\Delta = -0.94$) further supports the assumption that *pytorch-base* tends to consume less energy, which might imply that it adopts better optimizations for ML workloads, in our case, for the ResNet model. Smaller values of Cliff's delta, such as 0.08 between *python-base* and *ubuntu-base*, indicate relatively minor differences in energy consumption, suggesting these two base images exhibit similar resource usage. In general, the analysis highlights substantial differences in energy efficiency between the various base images, with *cuda-base* and *pytorch-base* showing to be more energy-efficient choices for ResNet workloads.

For the baseline, we conducted the experiment using all five Docker images in an idle state. As shown in Figure 5, there is no significant difference in energy consumption variation among the images. While *cuda-base* and *python-base* exhibit slight decreases in energy consumption, these changes are marginal. Similarly, Figure 6 reveals that there is no noticeable difference in the time taken to run the idle stage. This suggests that, although the Docker images have similar start-up times, any noticeable differences in performance or energy consumption are likely to emerge during the execution of the workload. We also hypothesize that different workloads may introduce further variations in the results. Therefore, for future work, a more in-depth analysis of the performance and energy consumption across various workloads is needed.

## 5 Discussion

In this section, we focus on explaining the results and answer *RQ2: "What are the configurations in Docker images that affect energy consumption the most?"*. The results of the experiment show an indication that the Docker image configuration can have a significant impact on the energy consumption on the Machine Learning workload described in Section 3, even when the runtime libraries installed on them are apparently identical. Testing has shown that, in general, more tightly integrated containers exhibit lower energy consumption than those on which manual installations were performed. However, not all optimized containers performed as expected. In this section, we discuss the possible reasons and implications of these results.

### Environment Variables

Environment variables are one of the less obvious differences between images that can affect performance. Taking a closer look, our version of PyTorch CUDA sets the *PYTORCH_CUDA_ALLOC_CONF* environment variable to *expandable_segments:True*. This is a memory management setting that tells PyTorch to use larger memory segments that can grow as needed, as opposed to allocating and freeing smaller memory segments [13]. The use of larger segments means less frequent memory operations, hence lower energy consumption.

### Pip and Conda

PyTorch CUDA uses Conda[9] to manage dependencies, which is a system package manager. The other images use Pip [10], which can only install Python libraries.

Conda solves the full dependency tree and install all the needed libraries in the CUDA toolkit, installing precompiled binaries for the target platform. Pip also only bundles CUDA runtime [11], which is more simplified than the full driver and might result in a loss of performance. Combined with a possible mismatch in the native libraries that would make PyTorch may fall back to CPU or other kernels.

### Dynamic Linking Overhead

There are two ways to link libraries to source code. On one hand, there is static linking, which is most often performed immediately after compilation and results in a self-contained executable. On the other hand, dynamic linking is performed during run time and uses shared libraries available on the system [14]. Dynamically linked programs usually use so-called *trampolines* to resolve the memory addresses of library functions at runtime and this can result in a higher instruction count and more branching operations [15].

It is possible that the optimized images will have statically-linked binaries compiled by the maintainers, while images with manual installations use less integrated dynamic linking.

### Balancing Image Size

Table 6 shows a summary comparison of *pytorch-base* with all of the other images in terms of consumption and image size. We can see that *nvcr-base* is disproportionately larger than the other images, probably due to its large-scale enterprise applicability. Although image size does not directly affect consumption during inference, it affects consumption during download and increases size requirements. When scaled to a large number of servers, this can increase costs and even offset the potential reduction in inference energy consumption.

In the context of this study, one potential explanation for the relatively small difference in energy consumption of the enterprise container is that it has been designed for workloads that far exceed the scope of the one we have tested. However, the result shows that images should be chosen with workload scale in mind.

| Image | Consumption Diff(%) | Size Diff (%) |
|---|---|---|
| cuda-base | 454.88 (1.22%) | 3.29 (34.2%) |
| nvcr-base | 1003.72 (2.69%) | 31.24 (324.74%) |
| python-base | 1084.84 (2.9%) | -0.51 (-5.3%) |
| ubuntu-base | 1131.2 (3.03%) | 0.33 (3.43%) |

Table 6: Relative with *pytorch-base*, in terms of median energy consumption (J) and image size (GB)

[9] https://anaconda.org/anaconda/conda

[10] https://pypi.org/project/pip/

[11] https://docs.nvidia.com/cuda/cuda-runtime-api/driver-vs-runtime-api.html

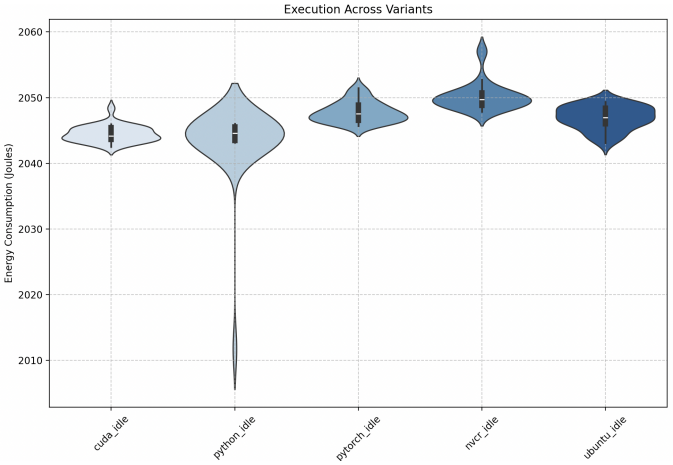


Figure 5: Baseline: Energy consumption during in idle state.

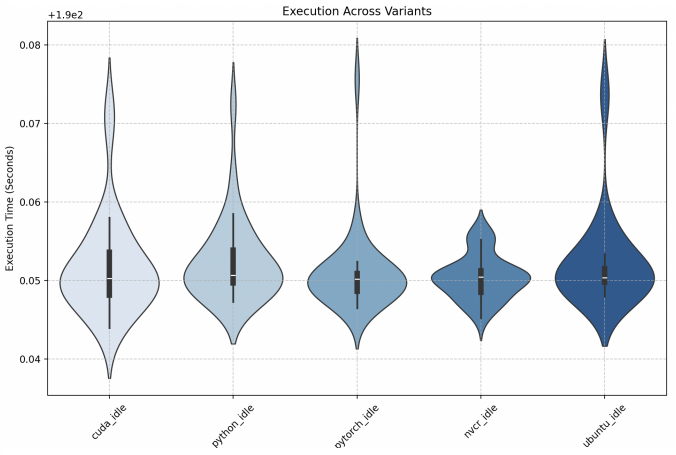


Figure 6: Baseline: Execution time in idle state.

### 5.1 Threats to validity

When conducting an experiment, potential risks that could affect the validity of the results have to be assessed. Three limitations could have been considered during the study. To some extent the time taken for running the experiments could prevent from revealing more significant differences between the set ups of the images. While an experiment takes around 14 minutes, a short amount of time, performing it 30 times increases the overall time and forces us to limit the run time of it. Secondly, the startup costs of a Docker container were taken into consideration when measuring the consumption, leading to a potential disruption of the results. However, in a real-world scenario, the workloads would be long running, and therefore, the startup costs become insignificant. The experiments were run on a single machine with the hardware stated in Section 3.1. Nonetheless, this set up is not representative of every machine. The limited size of RAM (i.e. system memory) and VRAM (i.e. video RAM) can lead to the usage of swap memory (i.e. when primary memory is insufficient, data overflows from one type of memory to another), thus increasing the energy consumption. Moreover, our experiments use a consumer-grade GPU that lacks the tensor cores of the current generation. This means that the energy consumption reported is not representative of the current generation hardware. Furthermore, the NVIDIA PyTorch NGC base image is optimized for enterprise-grade hardware like DGX systems, which means that our hardware did not leverage them.

### 5.2 Future work

The results obtained in this study indicate that container design choices have energy consumption implications. There are multiple research directions for further exploration of this problem, and we propose some in this subsection.

To help generalize the findings, the workloads can be run across a wider spectrum of hardware (e.g., different GPU generations). This can determine if the consumption difference is a trend on all hardware setups or is only limited to specific components. Building on the same idea, other frameworks (e.g., TensorFlow, JAX) should be explored as well to see if the same trends can be applied to them.

In the area of container improvement, the effect of individual Dockerfile layers can be isolated and tested to see their effect on performance and energy. A very interesting idea to explore after determining the effects of individual layers would be to extend tools like *dive*[12], that now focus on optimizing container size, to also support automatic image auditing for energy efficiency. This could help make energy issues visible and enable developer action.

## 6 Conclusions

To conclude, this study measured the energy consumption of five different Docker images during the inference phase of a Machine Learning workload. The results indicate that image configuration can have a significant impact on energy consumption: ready-made, integrated images exhibit, in general, lower consumption patterns than manual installations. However, they also show that more integration is not always better, and the base image for deployment should be chosen with the scale of the workload in mind.

Further research can be conducted to generalize the finding and explore the impact of Docker layers on energy consumption and create tools to help developers understand these effects and help them optimize the energy consumption of their containers.

## References

[1] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp, 2019.

[2] Roberto Verdecchia, June Sallou, and Luís Cruz. A systematic review of green ai, 2023.

[3] Radosvet Desislavov, Fernando Martínez-Plumed, and José Hernández-Orallo. Trends in ai inference energy consumption: Beyond the performance-vs-parameter laws of deep learning. *Sustainable Computing: Informatics and Systems*, 38:100857, April 2023.

[12] https://github.com/wagoodman/dive

[4] Sailesh Oduri. Revolutionizing machine learning model serving with containerization. *JOURNAL OF ADVANCED RESEARCH ENGINEERING AND TECHNOLOGY (JARET)*, 3(1):67–75, 2024.

[5] Eddie Antonio Santos, Carson McLean, Christopher Solinas, and Abram Hindle. How does docker affect energy consumption? evaluating workloads in and out of docker containers, 2017.

[6] Mehul Warade, Kevin Lee, Chathurika Ranaweera, and Jean-Guy Schneider. Monitoring the energy consumption of docker containers. In *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 1703–1710, 2023.

[7] Enrique Barba Roque, Luis Cruz, and Thomas Durieux. Unveiling the energy vampires: A methodology for debugging software energy consumption, 2024.

[8] Bailey Tjiong. The impact of base image selection on the energy efficiency of containerized applications in docker, 2023.

[9] Negar Alizadeh and Fernando Castor. Green ai: a preliminary empirical study on energy consumption in dl models across different runtime infrastructures. In *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for AI*, CAIN 2024, page 134–139. ACM, April 2024.

[10] Soyeon Park and Hyokyung Bahn. Performance analysis of container effect in deep learning workloads and implications. *Applied Sciences*, 13(21), 2023.

[11] Raluca Maria Hampau, Maurits Kaptein, Robin van Emden, Thomas Rost, and Ivano Malavolta. An empirical study on the performance and energy consumption of ai containerization strategies for computer-vision tasks on the edge. In *Proceedings of the 26th International Conference on Evaluation and Assessment in Software Engineering*, EASE '22, page 50–59, New York, NY, USA, 2022. Association for Computing Machinery.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[13] Cuda semantics: Optimizing memory usage wth pytorch_cuda_alloc_conf. "Accessed 29-03-2025".

[14] Michael Franz. Dynamic linking of software components. *Computer*, 30(3):74–81, March 1997.

[15] Varun Agrawal, Abhiroop Dabral, Tapti Palit, Yongming Shen, and Michael Ferdman. Architectural support for dynamic linking. *SIGPLAN Not.*, 50(4):691–702, March 2015.