Energy-Efficient ML Workloads in Docker: Measurement and Evaluation using EnergiBridge

1st Ahmed Ibrahim
Computer Science Department
Delft University of Technology
Delft, Netherlands
aibrahim3@tudelft.nl

2nd Ahmed Driouech
Computer Science Department
Delft University of Technology
Delft, Netherlands
adriouech@tudelft.nl

3rd Taoufik El Kadi
Computer Science Depathartment
Delft University of Technology
Delft, Netherlands
tkadi@tudelft.nl

4th Moegiez Abbaas Bhatti
Computer Science Department
Delft University of Technology
Delft, Netherlands
moegiezbhatti@tudelft.nl

***Abstract*—As machine learning workloads grow, their energy use is becoming a major concern. AI systems are expected to consume up to 73.9 TWh in the next decade [1], contributing to the rising footprint of the ICT sector, which now accounts for nearly 10% of global electricity use [2]. Docker is widely used in ML workflows for its portability and reproducibility [3], but the energy impact of different container configurations remains unclear. This paper presents a framework using EnergiBridge [7] to measure and compare the energy consumption of common Docker images and their CPU-optimized variants across three ML models and operating systems. The results show that optimized containers significantly reduce energy use in computationally intensive models, with consistent gains across platforms. The complete framework is available as an open source in [4].**

***Index Terms*—Docker, Machine Learning, EnergiBridge, Energy Measurement, Sustainability, Containerization**

I. Introduction

The energy consumption from artificial intelligence and machine learning is becoming a growing concern. Recent estimates suggest that AI workloads could consume up to 12.214 TWh in just five years and as much as 73.891 TWh over the next decade [1]. These figures show the long-term energy burden of AI systems, especially as they continue to scale. This trend is part of a wider increase in the energy footprint of the ICT sector, which now represents nearly 10% of global electricity consumption and is still increasing [2]. As AI applications continue to expand, energy efficiency in their development and deployment will play a great role in their sustainability and environmental impact.

At the same time, containerization technologies have become standard in ML workflows. Docker, in particular, has seen a fast growth in usage in development environments [3]. Its main advantages, namely reproducibility, portability, and isolation, make it a suitable tool for packaging and deploying ML models. Containers allow teams to ensure consistency between different systems with fewer configuration issues. Despite their widespread use, Docker images are rarely evaluated for energy performance. Even when running identical workloads, differences in the base image, system libraries, and thread configurations can lead to varying levels of energy consumption. For developers aiming to build more sustainable systems, there is currently little guidance or visibility into how these container-level choices affect power usage. This makes it difficult to assess the environmental impact of deployment decisions and limits the potential for energy-aware development practices.

To address this gap, we propose a framework that uses EnergiBridge[1] to systematically measure, compare and report the energy consumption of various Docker images used for ML tasks. The aim is to help developers make better informed and sustainable choices by:

1) Using EnergiBridge for energy profiling, as it offers high accuracy, detailed logging, and compatibility across different platforms.
2) Measuring the energy consumption of Docker images during key ML tasks, such as model training and inference.
3) Comparing widely used Docker base images to identify the most energy-efficient options.
4) Automating the energy measurement process within Docker containers to ensure results are reproducible and consistent.
5) Providing intuitive and detailed visualizations, such as violin plots and box plots, to illustrate energy consumption differences.
6) Performing statistical validations to confirm the reliability and significance of the findings.
7) Releasing the solution as an open-source project to encourage transparency, community collaboration, and promote sustainable software practices.

[1]https://github.com/tdurieux/EnergiBridge/blob/main/README.md

A. Research questions

This study is guided by the following research questions:

1) How does the choice of Docker base image affect the energy efficiency of ML workloads?
2) Do CPU-optimized Docker images offer measurable energy savings compared to standard base images?
3) Are these differences statistically significant across different operating systems and ML models?

To explore these questions, we evaluated three common operating systems (Ubuntu, Debian, Fedora) and their CPU-optimized counterparts using three representative ML workloads. Our methodology ensures consistent, reproducible measurement and applies statistical analysis to validate findings. By releasing the framework as an open source, we aim to promote sustainable practices in the design of ML infrastructure.

The remainder of this paper is organized as follows: Section II discusses the background information and Section III the related work . Section IV presents the experimental methodology. Section V reports the results and analysis. Section VI discusses the findings and implications. Section VII concludes with future directions. Finally, a link to the Github repository can be found in the references [4].

II. Background

A. Docker and Containerization

Docker has emerged as a leading platform for containerization, offering lightweight virtualization possibilities. By encapsulating applications and their dependencies within isolated environments, called containers, Docker ensures consistency across a great variety of deployments settings. However, this convenience does have an impact on system performance and energy consumption. Studies have shown that running applications within these containers can lead to increased energy consumption when running test in Docker, which can usually be assigned to performance overhads associated with I/O system calls [5], [6].

B. Energy Measurement using EnergiBrigge

In order to measure the energy consumption of Ml workloads with different docker image configurations, a reliable energy profiling tool is needed. Traditional utilities like Intel Power Gadget and pwerTOP provide some visibility into system level power usage, but their limitations in cross platform compatibility and hardware support limit the use in different deployment scenarios.

To address these challenges, we use EnergiBridge [7]. This is an open-source framework for cross-platform energy measurements. Supporting a wide range of operating systems, including Linux, Windows, and macOS, and being compatible with various CPU architectures such as Intel, AMD, and Apple ARM. It captures system resource usage during command execution, outputting data in a structured format. This makes it ideal for integration with automated workflows.

III. Related Work

Several recent studies have investigated the energy efficiency of containerized environments in the context of AI and other computationally expensive applications. Silva de Souza et al. introduced Containergy, a profiling tool that evaluates both performance and energy use of containerized workloads [8]. Their results show that software configuration plays an important role in energy consumption, with some models leading to more than 300% energy differences depending on how they are run. The study highlights that profiling tools can help developers identify energy-saving opportunities during development and deployment.

Santos et al. compared the energy usage of Docker containers with bare metal Linux environments in multiple applications [5]. Their results revealed a consistent increase in energy consumption when workloads were executed in containers. This overhead was mostly attributed to differences in how I/O system calls are handled in containerized setups. Although Docker's performance is often considered close to native, the study shows that energy efficiency can be affected depending on the nature of the workload.

Hampau et al. examines how different AI containerization strategies influence energy and performance outcomes on edge devices [9]. Their work tested Docker, WebAssembly, and ONNX Runtime on Raspberry Pi hardware using computer vision tasks. The results showed that Docker was not always the most energy-efficient solution, particularly on constrained devices. Alternatives like WebAssembly often outperformed Docker in both speed and energy use.

While these studies provide insights into container energy behavior, they either focus on comparing Docker with non-containerized environments or explore broad container strategies without examining the role of Docker image configurations themselves. Little work has been done to understand how different Docker base images, or their optimized variants, influence energy use during ML training and inference. This project addresses that gap by isolating the impact of Docker image configuration across multiple workloads and operating systems, using a reproducible setup.

IV. Methodology

A. Experimental Setup

To assess the energy consumption of the Docker images, we ran experiments using EnergiBridge. Experiments were conducted on a Macbook Pro with an Apple M3 chip (8-core CPUL 4 performance, 4 efficiency cores, up to 4.05 GHz), 16 GB LPDDR5 RAM, and a 512 GB SSD. The system ran macOS Sonoma 14.2 and was powered via a 70W USDB-C adapter on a 230V AC source with the bathery fully charged. The container execution was facitilated by Docker Desktop 4.25.0 on the ARM64 architecture. MacOS was set to "High Power" mode and background processes were minimized.

B. Docker Image Selection

We selected six Docker images: three base images(Ubuntu 22.04, Debian 12, Fedora 38) and three CPU-optimized vari-

ants. We chose for this set of images because of the need to evaluate energy consumption across different, realistic ML deployment scenarios while using the MacBook M3's ARM64 architecture. The base images used Ubuntu 22.04, Debian 12 and Fedora 38 as operating system (OS) and were chosen to represent widely adopted Linux distributions in ML workflows. Each OS has its own distinct design philosophies and characteristics. Ubuntu 22.04 is favored for stability, extensive documentation and adoption in ML environments. Debian 12 is a more lightweight OS with a robust foundation while Fedora 38 is a modern feature-rich environment often used in development settings. Each base image was configured with Python 3.10 and essential ML libraries installed in a virtual environment. These libraries were selected for the usage in ML projects in order for us to mimic typical development setups. The base image serve as standard un-optimized configurations that developers might deploy without specific performance tuning. The CPU-optimized variants extended these base images to explore energy consumption with specific enhancements. We incorporated libraries like OpenBLAS, libop-dev and libatlas-base-dev, which provide optimized multi-thread operations. OpenBLAS accelerates matrix operations in NumPy and PyTorch while libop-dev enables parallelization via OpenMP. We set specific environment variables to cap thread counts at 4 and align with the M3's performance core count to prevent overloading the efficiency cores.

C. ML Workload Configuration

Three ML models were tested: (1) a customer neural network for binary classification, (2) a fine-tuned DistilGPT2 for language modeling, and (3) a RandomForest classifier. The neural network consists of a 10-layer PyTorch model with ReLU and dropout, which was trained for 1000 epochs on a preprocessed dataset with SMOTE oversampling and standardization. DistilGPT2 was fine-tuned for 2 epochs on a 10-sentence dataset about sustainable software engineering with a batch size of 2. The RandomForest is optimized using GridSearchCV and trained on the same dataset as the neural network. Training durations varied across the models.

D. Integration with EnergiBridge

EnergiBridge [7] is an open-source cross-platform energy measurement tool, and provides support for various operating systems. EnergyBridge allows you to measure the energy consumption of any command ran in the terminal, thus for the purpose of this paper we used it to measure the energy consumption of our automated experiments. The tool can be used by running a command with the following format *"energibridge[.exe] [OPTIONS] [COMMAND]..."*. In our case specifically, *"*{*energiBridge_path*}*/target/release/energibridge -o* {*full_temp_path*} *–summary* {*docker_query*}*"*. The option *"-o* {*full_temp_path*} *–summary"* allows us to capture the results of the experiment in the csv file at *"full_temp_path"* after the command *"*{*docker_query*}*"* is executed.

E. Automation and Reproducibility

The experiment was fully automated and a replication package can be found on the github page. We wrote a python script that passes a docker query as a command to EnergiBridge. This docker query runs a selected python script ({*model_path*}) inside a selected docker image (*ml-energy-*{*image*}) which is pulled from DockerHub: *'docker run -it –rm -v "$(pwd):/app/repo" taoufikel/ml-energy-*{*image*}*:latest python /app/repo/*{*model_path*}*'*. In total there are 6 docker images and 3 python scripts, the experiment runs each combination 31 times with a 60 second cool-down in between. Note that the first run is a warm-up run. Every run is saved in a csv file in the results directory and named accordingly for later analysis. There is one major concern with reproducibility, as the automation is only tested on MacOS. Thus, the possibility to run on other operating systems still needs to be addressed.

V. Results

This section presents the results of the energy consumption analysis across different combinations of models, operating systems, and image types. For each model, a total of 31 measurements per combination were collected, resulting in 186 runs. Each run corresponds to a CSV file containing power measurements over time. From these, the total energy consumption in joules was computed by integrating the power values throughout the duration of the run.

The first step of the analysis involved identifying and removing erroneous and outlier data points. This is addressed in the first subsection, where descriptive statistics are presented for all combinations, both before and after filtering. The remaining subsections are organized by model and report the differences in energy consumption across the various Docker images used for that model. Each subsection presents the results for six images: one base and one optimized image for each of the three operating systems. These are visualized using time series plots that show the power usage across time, as well as violin plots that illustrate the distribution of total energy consumption per run. At the end of each subsection, statistical significance tests are reported to examine whether the observed differences between configurations are likely to be meaningful. This research uses an alpha value of p ¡ 0.05 to conclude statistical significance.

A. Outlier Detection

Before analyzing the data, erroneous or incomplete files were removed. Subsequently, we applied an outlier detection method based on the interquartile range (IQR). A value was considered an outlier if it was more than 1.5 times the IQR below the first quartile or above the third quartile. This filtering step was performed for each unique combination of model, operating system, and image type. Table I reports the descriptive statistics of energy consumption, both with and without outliers. It includes the mean, median, and standard deviation, as well as the number of outliers removed for each group. For each statistic, the value with outliers is shown on

top, and the value after filtering is shown directly below.

The number of outliers varied across the different combinations. In total, `llm` and `nn` had noticeably more outliers than `rf`, with the latter showing at most two per group and in most cases only one. The most extreme cases appeared in the `nn` model, where several combinations had five or more outliers. The `ubuntu` operating system accounted for the highest number of outliers overall, followed by `debian`. In contrast, `fedora` consistently produced more stable distributions with fewer outliers between models. There was no strong pattern in terms of image type, as both base and CPU-optimized configurations produced outliers in comparable numbers.

In almost all combinations, outliers were associated with higher energy consumption. This was especially clear for the `nn` model, where the unfiltered means were often inflated by extremely large values. The same pattern appeared in several `llm` combinations, although the differences were less pronounced. Removing these outliers resulted in lower means and reduced standard deviations, which made the distributions more comparable and less affected by rare deviations. This can be observed in Table I, where the stacked values in each cell show a clear drop in both the central tendency and the spread after filtering.

B. Docker Image Configurations – Large Language Model

This subsection presents the energy consumption results for the large language model across six different Docker image configurations. For each operating system, two configurations were tested: a base image and a CPU-optimized image. The comparison focuses on average power usage over time and the total energy consumed per run.

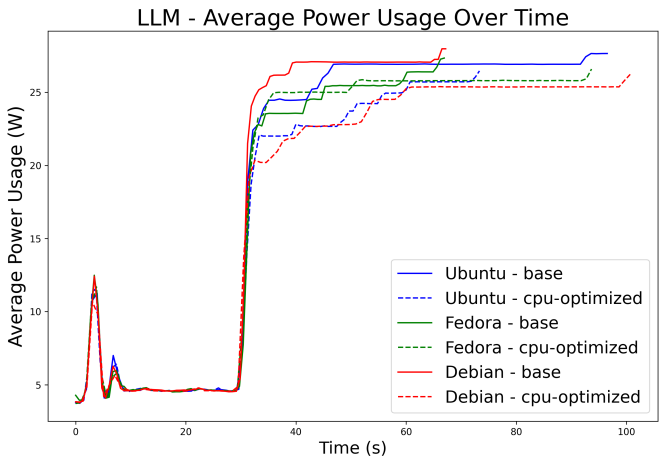

Fig. 1. Average power usage over time for the large language model across six Docker configurations.

Figure 1 shows the average power usage over time for each of the six configurations. Each curve represents the mean of multiple runs, aligned along a common time axis. Differences between the base and CPU-optimized images are mostly visible in the second half of the run. In all three operating systems, the base image leads to higher sustained power usage compared to the CPU-optimized alternative. The most distinct gap is visible in `debian`, where the base image shows a consistently higher curve than its CPU-optimized counterpart. The lines for `ubuntu` and `fedora` are closer together, but the same general trend is observed.

Figure 2 presents the total energy consumption per run using violin plots. For each operating system, two distributions are shown: one for the base image and one for the CPU-optimized image. The violins illustrate the distribution shape, while the median is marked in the center. For all three operating systems, the CPU-optimized image results in lower median energy consumption. The largest difference is observed in `fedora`, where the CPU-optimized image produces a visibly narrower and lower violin. In contrast, `ubuntu` and `debian` show smaller gaps between the two image types.

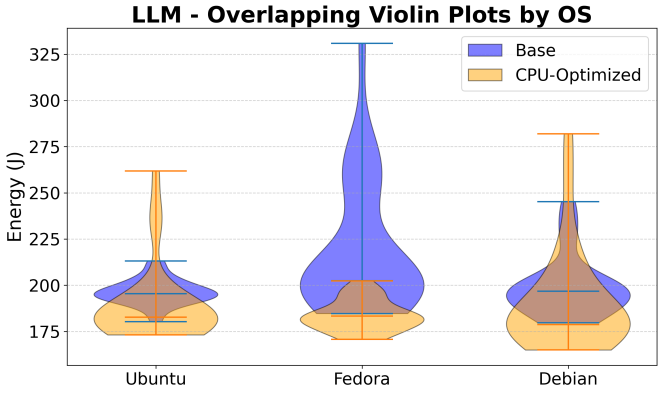

Fig. 2. Violin plots showing total energy consumption per run for each OS and image type (base vs CPU-optimized) for the large language model.

Statistical tests were performed to assess the significance of the observed differences. Table II reports the results. When comparing operating systems for the base image, both `ubuntu` and `fedora` differ significantly from `debian`, with p-values well below 0.01. No statistically significant differences were found between operating systems when using the CPU-optimized image. Finally, when comparing the image types within each OS, all three tests showed statistically significant differences, with `ubuntu` and `fedora` yielding the lowest p-values.

C. Docker Image Configurations - Neural Network

The figure below presents the average power usage over time for the neural network model across all six Docker configurations. As seen in Figure 3, the base images consistently show higher power usage across the entire duration of the run. In all three operating systems, the base image reaches a stable usage of around 23W, while the CPU-optimized image stabilizes closer to 21W. The differences appear consistent

TABLE I

DESCRIPTIVE STATISTICS OF ENERGY CONSUMPTION (IN JOULES) ACROSS ALL COMBINATIONS OF MODELS, OS, AND IMAGE TYPES. FOR EACH STATISTIC, THE VALUE WITH OUTLIERS IS SHOWN ON TOP, AND THE VALUE WITHOUT OUTLIERS IS SHOWN BELOW.

Model	OS	Image Type	Mean (J)	Median (J)	Std (J)	Number of Outliers
llm	debian	base	214.15	198.08	46.09	4
			199.10	196.81	16.57	
llm	debian	cpu-opt	219.61	184.24	77.27	5
			191.44	178.74	27.70	
llm	fedora	base	229.24	203.11	51.50	2
			219.34	202.51	35.38	
llm	fedora	cpu-opt	203.53	183.55	63.74	4
			183.71	183.29	8.26	
llm	ubuntu	base	230.42	197.65	81.91	7
			196.45	195.38	7.07	
llm	ubuntu	cpu-opt	224.16	184.06	112.20	5
			189.45	182.72	21.36	
nn	debian	base	561.72	456.62	215.83	7
			460.01	454.24	15.56	
nn	debian	cpu-opt	321.19	311.41	24.30	3
			314.83	311.20	10.66	
nn	fedora	base	532.43	465.48	184.87	5
			464.31	458.11	13.32	
nn	fedora	cpu-opt	2024.25	311.72	9498.36	4
			314.01	311.43	8.19	
nn	ubuntu	base	488.79	458.43	83.59	4
			460.23	455.82	9.71	
nn	ubuntu	cpu-opt	1963.61	313.02	9210.88	6
			314.13	312.40	7.27	
rf	debian	base	1046.30	1034.47	27.59	1
			1043.59	1034.16	23.47	
rf	debian	cpu-opt	2134.55	1037.25	6008.74	1
			1055.37	1037.00	31.21	
rf	fedora	base	1077.28	1062.56	34.16	2
			1071.42	1061.70	26.45	
rf	fedora	cpu-opt	3447.68	1067.74	13209.23	4
			1077.56	1067.74	24.94	
rf	ubuntu	base	1173.95	1118.42	230.46	1
			1132.98	1118.28	33.30	
rf	ubuntu	cpu-opt	1360.65	1121.55	1221.68	1
			1141.33	1121.42	37.62	

TABLE II

STATISTICAL TEST RESULTS FOR THE LARGE LANGUAGE MODEL. MANN–WHITNEY U TESTS WERE USED FOR ALL COMPARISONS.

Comparison	Type	p-value
Ubuntu vs Fedora (base)	OS difference	0.0017
Ubuntu vs Debian (base)	OS difference	0.9925
Fedora vs Debian (base)	OS difference	0.0026
Ubuntu vs Fedora (cpu-opt)	OS difference	0.7286
Ubuntu vs Debian (cpu-opt)	OS difference	0.4154
Fedora vs Debian (cpu-opt)	OS difference	0.8240
Ubuntu (base vs cpu-opt)	Image type	0.0001
Fedora (base vs cpu-opt)	Image type	0.0000
Debian (base vs cpu-opt)	Image type	0.0038

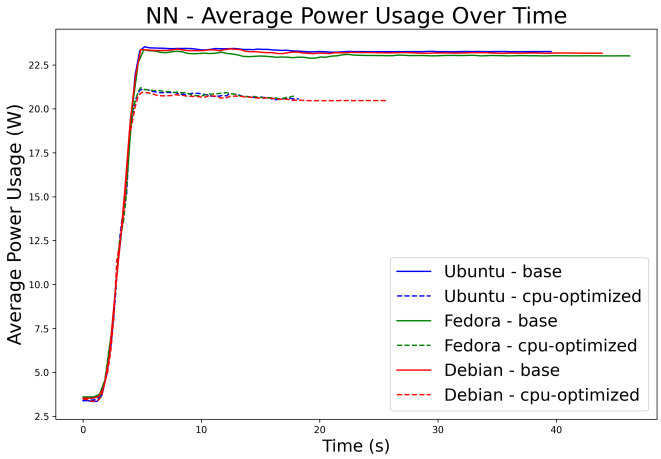

Fig. 3. Average power usage over time for the neural network across six Docker configurations.

throughout, without abrupt fluctuations. The two groups form clearly distinct layers, where the power usage of the CPU-optimized images remains below that of their base counterparts for the full length of the run.

This pattern is reflected in the total energy consumption shown in Figure 4. Across all three operating systems, the CPU-optimized image results in a lower and more compact energy distribution. The largest separation between image types is observed in `ubuntu`. The same holds for `fedora` and `debian`, although the difference is less in the latter two.

In all three cases, the distributions for the base image are wider and skewed towards higher energy use.

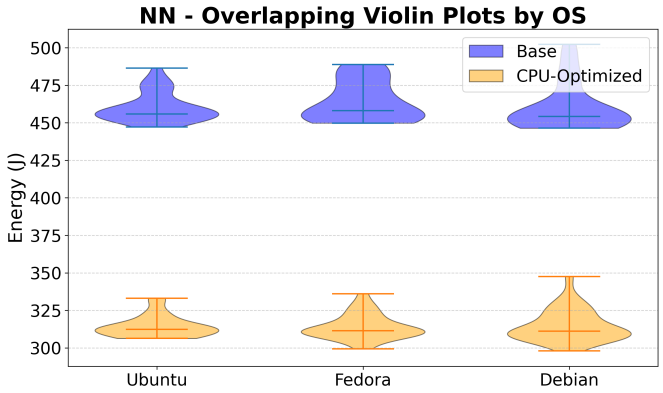

Fig. 4. Violin plots showing total energy consumption per run for each OS and image type (base vs CPU-optimized) for the neural network.

Statistical testing confirmed these visual findings. As shown in Table III, there were no statistically significant differences in energy consumption between the operating systems when using the same image type. For the base image, all p-values were above 0.09, and for the CPU-optimized image, they remained well above 0.6. In contrast, all comparisons between base and CPU-optimized images within the same operating system resulted in highly significant differences. The p-values for these tests were all below 0.001, confirming that the optimized image leads to a consistently lower energy footprint across the board.

TABLE III
STATISTICAL TEST RESULTS FOR THE NEURAL NETWORK MODEL. MANN–WHITNEY U TESTS WERE USED FOR ALL COMPARISONS.

Comparison	**Type**	p**-value**
Ubuntu vs Fedora (base)	OS difference	0.4933
Ubuntu vs Debian (base)	OS difference	0.1497
Fedora vs Debian (base)	OS difference	0.0944
Ubuntu vs Fedora (cpu-opt)	OS difference	0.7695
Ubuntu vs Debian (cpu-opt)	OS difference	0.6116
Fedora vs Debian (cpu-opt)	OS difference	0.6800
Ubuntu (base vs cpu-opt)	Image type	0.0000
Fedora (base vs cpu-opt)	Image type	0.0000
Debian (base vs cpu-opt)	Image type	0.0000

D. Docker Image Configurations – Random Forest

The figure below shows the average power usage over time for the random forest model. As seen in Figure 5, the difference between base and CPU-optimized images varies less across the three operating systems compared to the previous models. The differences between OS systems seem to be higher. The power curves for the `fedora` images are clearly higher than those for `ubuntu` and `debian`, both in base and CPU-optimized configurations. In `fedora`, the CPU-optimized image even surpasses the base image in average power usage. For `ubuntu` and `debian`, both configurations follow a similar pattern, with slightly lower usage for the optimized image towards the end of the run.

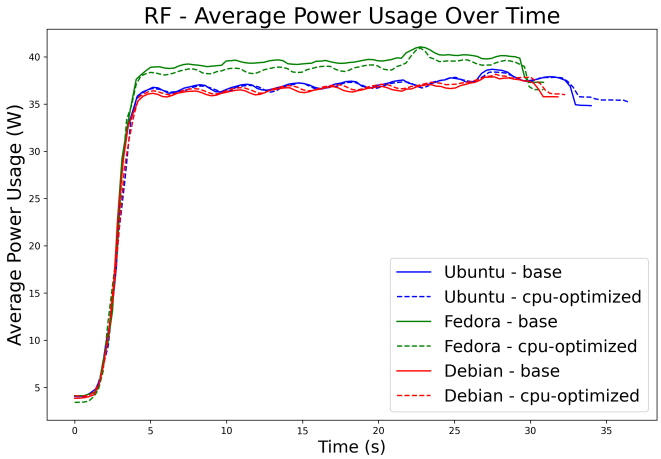

Fig. 5. Average power usage over time for the random forest model across six Docker configurations.

These observations are reflected in the violin plots shown in Figure 6. The distributions for the base and CPU-optimized images are close for all three operating systems. For `ubuntu`, the base image has a slightly lower median, while for `fedora`, the two distributions overlap almost completely. Only in `debian` does the optimized image result in a clearly higher median energy use, though the overall spread is small.

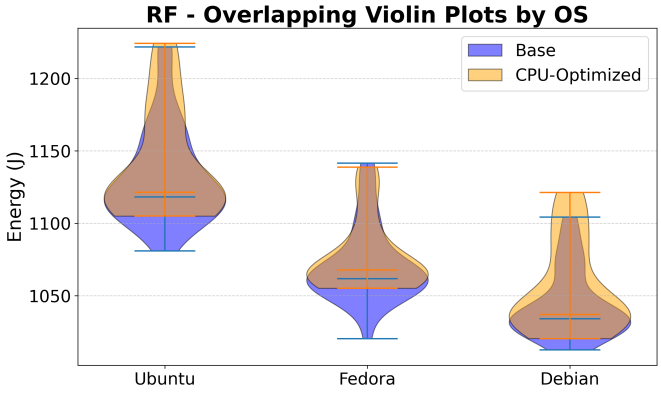

Fig. 6. Violin plots showing total energy consumption per run for each OS and image type (base vs CPU-optimized) for the random forest model.

The statistical tests in Table IV confirm that differences between operating systems are significant for both image types. All comparisons yield p-values below 0.001, suggesting that OS choice affects energy consumption more than image type for this model. When comparing base and CPU-optimized images within the same OS, only `debian` shows a statistically significant difference, with a p-value of 0.0468. The differences in `ubuntu` and `fedora` are not statistically significant.

TABLE IV
STATISTICAL TEST RESULTS FOR THE RANDOM FOREST MODEL. MANN–WHITNEY U TESTS AND WELCH'S t-TESTS WERE USED WHERE APPROPRIATE.

Comparison	Type	p-value
Ubuntu vs Fedora (base)	OS difference	0.0000
Ubuntu vs Debian (base)	OS difference	0.0000
Fedora vs Debian (base)	OS difference	0.0001
Ubuntu vs Fedora (cpu-opt)	OS difference	0.0000
Ubuntu vs Debian (cpu-opt)	OS difference	0.0000
Fedora vs Debian (cpu-opt)	OS difference	0.0009
Ubuntu (base vs cpu-opt)	Image type	0.3042
Fedora (base vs cpu-opt)	Image type	0.1490
Debian (base vs cpu-opt)	Image type	0.0468

VI. DISCUSSION

A. Insights

The results show that CPU-optimized Docker images can reduce energy consumption for machine learning workloads, particularly for models that rely on repeated numerical computations. In both the neural network and large language model tasks, the CPU-optimized containers consumed less energy across all three operating systems. These differences were statistically significant and consistent across multiple runs, indicating that container tuning using multithreaded libraries and controlled thread environments can lead to energy savings. The findings support the idea that energy efficiency can be improved without changing the underlying machine learning code or the hardware setup, but instead by selecting more suitable base images and library configurations.

The strongest improvements were observed in the neural network model, where the power usage for the optimized images was consistently lower across time and across all operating systems. The gap between the base and optimized images was clear and stable throughout the runs. This is likely due to the structure of the neural network, which involves a high number of matrix multiplications and benefits from optimizations in libraries such as OpenBLAS [10]. The CPU thread cap that was applied also aligned well with the number of performance cores on the test device, which may have helped reduce unnecessary load on the efficiency cores and avoid energy spikes.

The large language model followed a similar trend. While finetuning this model was less expensive and the differences in energy consumption were less extreme than those of the neural network, the optimized containers still consumed less energy in all tested environments. This was most visible in Fedora, where the distributions were narrower and the median energy values were lower. The results for Ubuntu and Debian also showed clear improvements, although the visual differences were more subtle. These outcomes show that the benefit of CPU-optimized containers extends beyond only large or computationally intense models and can be useful in a broader range of workloads, especially when inference is repeated many times at scale.

The random forest model presented a different pattern. Here, energy consumption was not clearly affected using optimized containers. Both the time series plots, and the energy distributions showed minimal variation between base and CPU-optimized images. The only statistically significant difference was found in Debian, and even there the overall gap was small. These results suggest that for workloads driven by tree-based logic and control flow, CPU-level mathematical optimizations have limited impact. Instead, energy differences were more strongly tied to the operating system. Fedora consistently used more power for this model, across both image types. This may be due to differences in how system resources are managed at the OS level.

In contrast, the neural network and language model tasks showed almost no statistically significant differences in energy use between operating systems when using the same image type. Across Ubuntu, Fedora, and Debian, energy usage stayed within a narrow range for each model-image pairing. This outcome suggests that once a consistent container environment is defined, the operating system has limited influence on energy consumption, at least for the types of ML workloads tested. From a practical perspective, this increases flexibility in deployment, as container optimization can yield similar energy improvements across different platforms without the need to standardize the host OS.

Taken together, the findings suggest that energy efficiency in ML workflows can be improved by paying attention to container configurations and the nature of the workload. For models that depend on matrix computations, replacing general base images with optimized alternatives offers significant savings. The fact that these gains were stable across different operating systems makes this approach widely applicable. For tasks where the underlying algorithm is less dependent on numerical libraries, container optimization may offer limited gains, and other parts of the system stack may play a larger role. As machine learning systems are deployed more widely [11], and the use of docker images increases [3], these small optimizations can make a meaningful difference at scale, especially in settings where resources are shared, or energy usage is monitored.

B. Challenges and Limitations

One of the main challenges in this project was the long runtime of the experiment. The machine used for testing was a personal laptop, not a dedicated benchmarking server. This limited the level of parallelization and placed constraints on how many configurations could reasonably be tested. Running the full set of model–OS–image combinations already took more than 30 hours, and any attempt to test additional operating systems, container configurations, or more complex models would have exponentially increased this time. Each new factor multiplies the total number of combinations, making it difficult to explore a wider configuration space within the limited project period.

The relatively modest computational power of the machine also influenced the selection of the model. For the large language model, we used a lightweight variant (DistilGPT2) and only performed minimal finetuning. Heavier models or

more extensive tuning routines were not feasible due to the time and memory requirements. This limits the generalizability of the results for larger-scale natural language processing tasks, where different models may exhibit different energy usage patterns. It is likely that more powerful hardware would not only reduce runtimes, but also allow testing of a broader set of workloads and more representative finetuning scenarios.

Our Docker image selection faced some limitations and constraints. We initially intended to use Alpine Linux as a base image. However, we faced compatibility issues with the ARM 54 architecture of the MacBook M3, as many ML libraries lacked prebuilt ARM64 binaries for Alpine's musl libc. This led to build failures and we excluded them from our study. We also explored GPU-optimized images that use CUDA to assess energy efficiency with GPU. While this worked for Ubuntu 22.04 using NVIDIA's official CUDA tags, Fedora 38 and Debian 12 lacked available NVIDIA tags for ARM64. Custom build was impractical for the time frame.

Despite these constraints, the findings still provide useful information on how container choices affect energy consumption. A more powerful benchmarking setup, extended test window, and broader hardware support would allow a deeper exploration of these effects in future work. For now, the current setup demonstrates that even modest adjustments to the container configuration can yield meaningful differences in energy efficiency.

VII. Conclusion and Future Work

The research has shown that CPU-optimized Docker containers can significantly reduce energy consumption for machine learning workloads, which is especially true for models with more intensive numerical computations, such as neural networks and language models. Using EnergiBridge, responsible and fine-grained measurements were taken. After analyzing the results, it became clear that the optimized container configurations consistently outperformed the base configurations in terms of energy efficiency, regardless of the underlying operating system.

However, several limitations reduced the scope of the study. Due to time and hardware constraints, the experiments were limited to lightweight models and a small set of container configurations. Additionally, automation and reproducibility were tested only on macOS, which could potentially limit portability to other platforms. Compatibility issues with Alpine Linux and ARM64 also prevented the inclusion of certain lightweight or GPU-accelerated configurations.

Future work should therefore focus on expanding the framework to support more powerful benchmarking environments and a wider range of ML models, operating systems, and hardware architectures, including GPU-based workloads. This would not only improve the ability to generalize the results, but would also enable more large-scale evaluation scenarios. Furthermore, integrating low-level profiling of system calls or library-specific energy usage could help uncover the precise sources of inefficiency. Applying the framework to cloud and edge environments would also offer valuable insights into trade-offs and sustainability challenges in real-world deployment contexts.

REFERENCES

[1] Ljubiša Bojić, Karlo Bala, Milovan Medojević, and Max Talanov, "AI and Energy Consumption: Social Aspects," *2024 9th International Conference on Smart and Sustainable Technologies (SpliTech)*, pp. 1–4, 2024, doi: 10.23919/SpliTech61897.2024.10612493.

[2] Erol Gelenbe, "The Measurement and Optimization of ICT Energy Consumption," *2022 IEEE International Symposium on Technology and Society (ISTAS)*, vol. 1, pp. 1–6, 2022, doi: 10.1109/ISTAS55053.2022.10227107.

[3] John Kreisa. Docker Index: Dramatic Growth in Docker Usage Affirms the Continued Rising Power of Developers. *Docker Blog*, July 2020. Accessed: 2025-04-03. Available at: https://www.docker.com/blog/docker-index-dramatic-growth-in-docker-usage-affirms-the-continued\--rising-power-of-developers/.

[4] Moegiez Abbaas Bhatti, Ahmed Ibrahim, Ahmed Driouech, and Taoufik El Kadi. Sustainable ML Workloads – EnergiBridge Docker Experiments. *GitHub Repository*, 2025. Accessed: 2025-04-04. Available at: https://github.com/almoegbhat2/sustainable-2.

[5] E. A. Santos and C. McLean, "How does Docker affect energy consumption? Evaluating workloads in and out of Docker containers," *Journal of Systems and Software*, vol. 146, pp. 14–25, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0164121218301456

[6] M. Warade, M. K. Nambiar, and A. S. K. Raju, "Monitoring the Energy Consumption of Docker Containers," in *Proceedings of the IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, Madrid, Spain, 2021, pp. 791–800. [Online]. Available: https://mehulwarade.com/files/papers/conference/COMPSAC-Docker_Energy.pdf

[7] J. Sallou, L. Cruz, and T. Durieux, "EnergiBridge: Empowering Software Sustainability through Cross-Platform Energy Measurement," arXiv preprint arXiv:2312.13897, Dec. 2023. [Online]. Available: https://github.com/tdurieux/EnergiBridge

[8] W. Silva-de-Souza, A. Iranfar, A. Bráulio, M. Zapater, S. Xavier-de-Souza, K. Olcoz, and D. Atienza, "Containergy—A Container-Based Energy and Performance Profiling Tool for Next Generation Workloads," *Energies*, vol. 13, no. 9, p. 2162, 2020, doi: 10.3390/en13092162.

[9] R. M. Hampau, M. Kaptein, R. van Emden, T. Rost, and I. Malavolta, "An Empirical Study on the Performance and Energy Consumption of AI Containerization Strategies for Computer-Vision Tasks on the Edge," in *Proc. 26th Int. Conf. on Evaluation and Assessment in Software Engineering (EASE '22)*, pp. 50–59, 2022, doi: 10.1145/3530019.3530025.

[10] Cristian Ramírez, Adrián Castelló, Héctor Martínez, and Enrique S. Quintana-Ortí. Performance Analysis of Matrix Multiplication for Deep Learning on the Edge. In *High Performance Computing. ISC High Performance 2022 International Workshops*, pages 65–76. Springer International Publishing, 2022. DOI: 10.1007/978-3-031-23220-6_5.

[11] Andrei Paleyes, Raoul-Gabriel Urma, and Neil D. Lawrence. Challenges in Deploying Machine Learning: A Survey of Case Studies. *ACM Computing Surveys*, 55(6):114:1–114:29, December 2022. Association for Computing Machinery, New York, NY, USA. DOI: 10.1145/3533378.