Greenlight: Highlighting TensorFlow APIs Energy Footprint

Saurabhsingh Rajput
Dalhousie University
Canada
saurabh@dal.ca

Maria Kechagia
University College London
UK
m.kechagia@ucl.ac.uk

Federica Sarro
University College London
UK
f.sarro@ucl.ac.uk

Tushar Sharma
Dalhousie University
Canada
tushar@dal.ca

ABSTRACT

Deep learning (DL) models are being widely deployed in real-world applications, but their usage remains computationally intensive and energy-hungry. While prior work has examined model-level energy usage, the energy footprint of the DL frameworks, such as TensorFlow and PyTorch, used to train and build these models, has not been thoroughly studied. We present Greenlight, a large-scale dataset containing fine- grained energy profiling information of 1284 TensorFlow API calls. We developed a command line tool called CodeGreen to curate such a dataset. CodeGreen is based on our previously proposed framework FECoM, which employs static analysis and code instrumentation to isolate invocations of TensorFlow operations and measure their energy consumption precisely. By executing API calls on representative workloads and measuring the consumed energy, we construct detailed energy profiles for the APIs. Several factors, such as input data size and the type of operation, significantly impact energy footprints. Greenlight provides a ground-truth dataset capturing energy consumption along with relevant factors such as input parameter size to take the first step towards optimization of energy-intensive TensorFlow code. The Greenlight dataset opens up new research directions such as predicting API energy consumption, automated optimization, modeling efficiency trade-offs, and empirical studies into energy-aware DL system design.

KEYWORDS

Energy measurement, Green Artificial Intelligence, Fine-grained energy measurement

1 INTRODUCTION

DL models have set state-of-the-art performance across several domains such as computer vision, natural language processing, and speech recognition [3, 5]. However, their extensive computational requirements for training and inference incur massive energy costs that continue to grow over time. For example, a recent study found that an image generation model consumed enough energy to fully charge an average smartphone to create a single image [13]. This level of energy intensity has raised serious concerns over the sustainability of increasingly large DL models. In fact, the computing needs of state-of-the-art models double approximately every 3.4 months [1]. Given these trends, it is required to improve their efficiency and limit the environmental impact of resource-intensive artificial intelligence [19].

The energy efficiency of DL-based systems can be improved by optimizing frameworks such as TensorFlow, PyTorch, and JAX, which are used to construct DL models. These frameworks expose APIs as building blocks that can be selected and composed in ways that profoundly affect model energy footprints. The same model can have vastly different energy costs depending on framework choice [6], hardware accelerators [4, 15], and the organization of its internal computations and data flows [10, 11].

For instance, Georgiou *et al.* [6] found that TensorFlow programs are generally more energy-efficient than PyTorch equivalents during the training stage, while PyTorch offers better energy efficiency during the inference phase. Optimizing energy consumption of a large DL model requires fine-grained energy profiles of their building blocks. Attributing energy consumption to specific APIs can help us isolate inefficiencies and, in turn, facilitate addressing them. API-level profiling provides the necessary "white-box" visibility to make informed optimizations that improve the energy efficiency of DL software.

Prior work on improving the energy efficiency of DL has focused primarily on model-level techniques such as pruning, quantization and knowledge distillation [7, 12]. However, there has been limited investigation into the software frameworks and API calls used to construct, train, and run these models. Existing studies lack a detailed examination of the energy footprint of common framework operations. While existing tools can provide coarse-grained energy measurements at the system or process level, fine-grained attribution and profiling at the API level is missing. To the best of our knowledge, no comprehensive dataset exists quantifying the energy consumption of individual DL framework API invocations. This hinders developers from making optimized API choices and developing energy-aware coding practices. For example, a developer using `tf.keras.Model.fit()`, which can consume over 100 Joules per epoch when training CNNs, may be unaware that switching to `tf.keras.Model.fit_generator()` can reduce energy usage by over 20% in certain contexts by eliminating redundant data processing. Our fine-grained profiling provides developers with the information needed to substitute energy-intensive APIs with more efficient alternatives depending on usage context.

Our work addresses the lack of fine-grained energy profiling for DL-based software through Greenlight, an API-level energy consumption dataset for TensorFlow. We developed a command line interface (CLI) tool that utilizes FECoM [16] to instrument TensorFlow code and measure the energy usage of API calls. We make the following contributions to the state of the art.

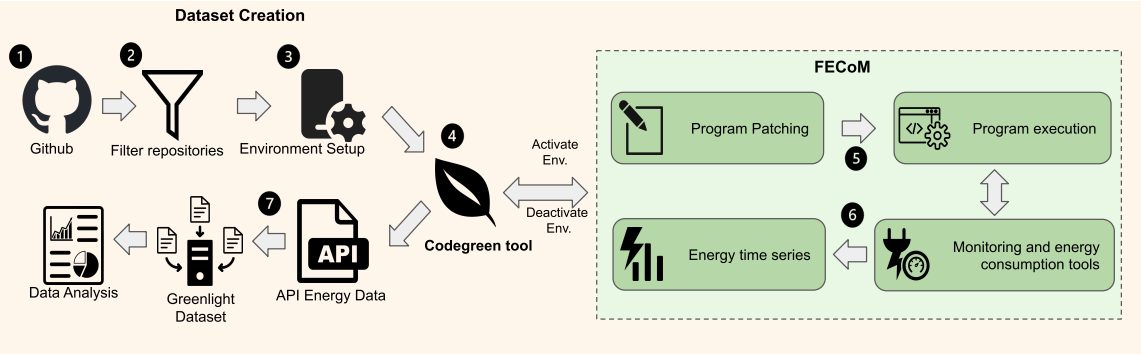

Figure 1: Dataset construction process

- **Greenlight**: A dataset containing 1,100 fine-grained energy measurements for TensorFlow APIs along with other relevant meta-data including execution time, timestamps, power draw time series, input argument, keyword and object size, hardware temperatures, Git commit info such as date, hash, and API call code location.
- **CodeGreen**: An easy-to-use CLI tool to profile the energy consumption of TensorFlow APIs.

Replication package: The proposed dataset Greenlight [18] as well as our CLI tool CodeGreen [17] are publicly available.

2 DATASET CONSTRUCTION

This section summarizes the mechanism that we adopt to identify and download the repositories and set up the environment to execute CodeGreen to obtain energy consumption data.

2.1 Downloading repositories

As shown in step 1 of Figure 1, we follow the steps mentioned below to identify and download repositories.

- First, we filter repositories using the GitHub Search API. In this step, we consider repositories with ≥ 100 stars and "tensorflow" in the title or owner name.
- We filter out repositories that have not been updated since Sept 2019 (TensorFlow 2 release) to focus on code using TensorFlow 2.
- We obtain 146 repositories covering a diverse set of domains, including computer vision, natural language processing, distributed computing, and so on.

2.2 Preprocessing

In step 2, we preprocess the selected repositories, adopting the following steps. We filter out repositories that do not come with the declared required dependencies, such as `requirements.txt`. Such declarations are critical to recreate execution environments. In this study, we keep our focus on Python notebooks as target scripts because they are standalone without the complexities of local dependencies. The final filtered set contains 564 Python notebook projects written using TensorFlow 2 APIs suitable for energy profiling targets.

2.3 Creating virtual environments

Once the final set of 564 target projects is identified, as shown in 3, we create isolated virtual environments for each project to install the required modules and dependencies, ensuring smooth execution. We utilize `requirements.txt` in each repo to install dependencies using the pip package manager into the virtual environment. This automated setup ensures properly configured virtual environments to run notebooks, making our experiments repeatable and replicable.

2.4 CodeGreen

Step 4 involves invoking our tool CodeGreen. It is a CLI tool that wraps our FECoM framework [16] to perform energy profiling of TensorFlow code. CodeGreen enhances FECoM by providing an easy installation via pip, a streamlined interface and automation for energy profiling workflows. The CLI abstraction makes utilizing FECoM's capabilities more accessible to developers. It operates in three key stages that we elaborate on below.

2.4.1 Patching. The tool CodeGreen leverages and extends a static instrumentation module of FECoM referred to as *Patcher* to isolate TensorFlow API calls within the code. Patcher parses the abstract syntax tree of the Python code and inserts wrapper code before and after each identified TensorFlow invocation. This wrapper code triggers the start and end of energy measurement for that API call. *Patcher* also extracts metadata such as function arguments, keywords, objects and their respective sizes, and execution times. CodeGreen enhances Patcher to capture GitHub metadata, expand support for profiling different argument types, and track the code line number of each API call.

2.4.2 Program execution. In step 5, CodeGreen executes the instrumented Python notebook within its configured virtual environment. The tool invokes the profiled TensorFlow APIs wrapped by the instrumentation code.

2.4.3 Energy measurement. As each instrumented API call executes, the inserted wrapper code triggers our FECoM framework to start and stop energy measurement as shown in 6. FECoM uses hardware performance counters provided by hardware vendors and exposed by the operating system to capture precise energy consumption data from the CPU, GPU, and RAM during API execution.

Ensuring machine stability and temperature is a critical aspect in this context. CodeGreen performs pre-measurement checks following the approach used in FECoM. This includes a temperature check that the CPU and GPU are below a threshold to maintain thermal stability. An energy stability check is also conducted to ensure CPU, RAM, and GPU power draw have low fluctuation, indicating no outside processes are interfering. Once both temperature and energy stability criteria are met, CodeGreen proceeds to execute instrumented scripts and collect measurements. These rigorous stability checks reduce noise and variability that could skew results, ensuring consistent conditions. To further improve robustness, each API call measurement is repeated five times. The mean energy is taken as the final measurement.

By automating the end-to-end process of instrumentation, execution, and fine-grained energy profiling, CodeGreen enables constructing API-level energy consumption dataset in a robust and reproducible manner.

2.5 Greenlight dataset description

2.5.1 Dataset overview. The Greenlight dataset provides fine-grained energy consumption measurements for 1284 TensorFlow API calls for 527 unique TensorFlow APIs across 564 open-source TensorFlow projects. The dataset contains a diverse range of operations, spanning layers, models, training, and other aspects of TensorFlow.

As a result of the energy profiling, the dataset incorporates comprehensive metadata for each TensorFlow API call, as shown in the example profile in Listing 1. This includes *execution time* of the API call, *timestamps* such as start and end times for the API call, *perf* measurement, and *nvidia-smi* sampling. Similarly, *Timeseries power draw data* is captured for CPU, GPU, and RAM during the API execution, *input argument size (in bytes)* are logged in for arguments, keywords and objects, *hardware temperatures* captures thermal context of CPU, GPU and RAM. Additionally, *git commit metadata such as date, hash-id, script path*, and API *call code location* is extracted for the API invocation. This metadata is combined with the energy measurements to generate detailed profiles for each TensorFlow API invocation.

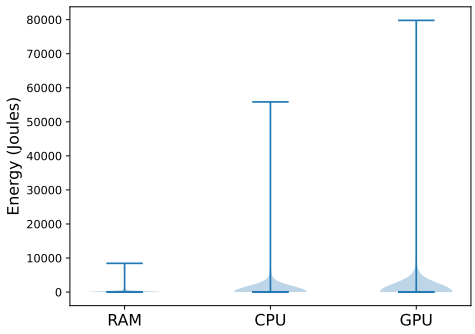

Figure 2: Energy consumption by hardware component

2.5.2 Dataset size and distribution. The Greenlight dataset contains a total energy consumption of 144, 067 Joules across the evaluated TensorFlow APIs. As seen in Figure 2, the GPU is the most energy-intensive component, consuming 79, 785 Joules (55%). The CPU uses 55, 843 Joules (39%), while the RAM uses 8, 439 Joules (6%).

The data exhibits high variability in energy consumption between operations as shown in Figure 2. The standard deviation of total energy per API call is 3, 429 Joules, which is 268% of the mean. This is likely due to differences in computational complexity and hardware utilization between operations.

```
{ "tensorflow.io.TFRecordWriter.close()":
    {
        "energy_data": {
            "cpu": "Power draw time series",
            "ram": "Power draw time series",
            "gpu": "Power draw and Temp. time series",
        },
        "times": {
            "start_time_execution": "Execution start time",
            "end_time_execution": "Execution end time",
            "start_time_perf": "perf start time",
            "end_time_perf": "perf end time",
            "sys_start_time_perf": "erf sys start time",
            "start_time_nvidia": "nvidia-smi start time",
            "end_time_nvidia": "nvidia-smi end time",
           "sys_start_time_nvidia": "nvidia sys start time",
          "begin_stable_check_time": "stability time stamp",
          "begin_temperature_check_time":"temp. time stamp"
        },
        "cpu_temperatures": "Temperature time series",
        "settings": {
            "max_wait_s": 120,
            "wait_after_run_s": 30,
            "wait_per_stable_check_loop_s": 20,
            "tolerance": 0,
            "measurement_interval_s": 0.5,
            "cpu_std_to_mean": 0.03,
            "ram_std_to_mean": 0.03,
            "gpu_std_to_mean": 0.01,
            "check_last_n_points": 20,
            "cpu_max_temp": 55,
            "gpu_max_temp": 40,
            "cpu_temperature_interval_s": 1
        },
        "input_sizes": {
            "args_size": "Size in bytes",
            "kwargs_size": "Size in bytes",
            "object_size": "Size in bytes"
        },
        "project_metadata": {
            "project_name": "Repo name",
            "project_repository": "https://github.com/...",
            "project_owner": "owner",
            "project_branch": "master",
            "project_commit": "Commit hash",
            "project_commit_date": "Time stamp",
            "script_path": "/home/...",
            "api_call_line": "Line number"
        }
    }
}
```

Listing 1: Schema of a TensorFlow API call in Greenlight

The dataset covers a wide range of energy intensity, from a minimum of 3.41 Joules for `tf.constant` to a maximum of 30,000 Joules for `tf.keras.Sequential.fit`. Optimizing the most intensive operations can provide significant energy savings.

The variability and range highlight the importance of fine-grained energy profiling. Subtle code changes can have an outsized impact on hardware efficiency. The GREENLIGHT dataset enables further analysis into the root causes of energy hotspots within TENSORFLOW workloads.

3 POTENTIAL RESEARCH APPLICATIONS

The GREENLIGHT dataset and CODEGREEN tool enable new research directions to build greener and more energy-efficient deep learning systems.

- **Predicting API energy consumption:** The dataset provides ground truth measurements not only including total energy consumed by an API but also other factors (such as passed parameters and their size). The captured information can be used to train machine learning models to predict the energy consumption of TENSORFLOW operations. Such an application can guide developers towards more efficient APIs.
- **Automated API optimization:** The energy profiles could also be utilized to automate the optimization of deep learning code, such as by auto-selecting API alternatives with lower energy or sequence optimizations.
- **Modeling energy-efficiency trade-offs:** Researchers can leverage the dataset to gain insights into trade-offs between model accuracy, performance, and energy consumption. This can guide the development of energy-aware models.
- **Enriching documentation:** The API-level energy data can be incorporated into TENSORFLOW documentation to promote energy-aware usage.
- **Education:** Educators can adopt CODEGREEN and the dataset into courses on deep learning and green software engineering to instill energy-aware coding habits.

4 RELATED WORK

Prior studies have examined the energy consumption of deep learning systems using coarse-grained measurements. Software tools such as *PowerTop* and *Perf* leverage hardware counters to profile system or process-level power but lack the granularity to attribute energy to fine-grained specific code [14, 20]. Physical power meters [9] enable accurate readings but require special equipment and measure energy only at the system level. Recent frameworks such as CODECARBON [2] and EXPERIMENT IMPACT TRACKER [8] estimate energy during model training but use sampling intervals > 10 seconds, which are too coarse for fine-grained analysis.

Research has also aimed to improve model efficiency through compression, quantization, and specialized training algorithms [7, 12]. However, model-centric techniques require retraining and cannot optimize existing models. Optimizing the surrounding TENSORFLOW software stack is an under-explored dimension. Prior work has lacked a detailed examination of the energy footprint of common TENSORFLOW ops used to build, train, and run deep learning models. Our work addresses this gap through GREENLIGHT, the first fine-grained API energy profiling dataset for TENSORFLOW. By isolating and measuring API calls through static instrumentation, GREENLIGHT establishes ground truth consumption for ops, revealing high variability across data sizes, hardware, and API sequences.

GREENLIGHT complements model-based approaches by enabling optimization of intensive TENSORFLOW code without changes to model architecture or hardware. It encourages energy-aware software development through detailed API-level profiling. By open-sourcing GREENLIGHT, we take a step towards sustainable and efficient AI systems.

5 THREATS TO VALIDITY

Internal Validity: Several factors could potentially affect the accuracy of the fine-grained energy measurements. Background operating system processes running on the machine introduce noise can skew results. We mitigate this by minimizing non-essential processes and subtracting out baseline idle energy. Additionally, we perform rigorous machine stability checks prior to measurement to reduce variability in factors like temperature and power draw that could impact results. To enhance reliability, we executed each experiment 5 times and automated the patching, profiling, and data collection. We provide detailed logs for replicability; the dataset and tool are hosted on open-source repositories.
External Validity: Our experiments were conducted on a fixed hardware configuration. Energy consumption is highly dependent on the underlying processor, GPU, etc. However, we measure and subtract baseline idle energy to improve hardware generalization. Analyzing multiple systems could further strengthen generalizability.

6 CONCLUSIONS, LIMITATIONS, AND FUTURE WORK

In this work, we present the GREENLIGHT dataset containing fine-grained energy profiling information for 1284 TENSORFLOW API calls across 564 projects. We also demonstrate an energy profiling tool, CODEGREEN. By providing an open-source tool for energy profiling and offering the first ever created energy consumption dataset for TENSORFLOW APIs, we open up many research possibilities.

However, our study has certain limitations. Specifically, the presented dataset focuses only on the TENSORFLOW framework. Also, experiments are limited to a single hardware configuration; analyzing the energy consumption of programs on multiple platforms would improve generalizability. In the future, we aim to expand the dataset from the framework, hardware, workload, and profiling technique aspects. We will perform empirical studies to uncover relationships between efficiency, code patterns and data properties. We plan to incorporate energy profiles into TENSORFLOW documentation to promote awareness. Finally, we hope to adopt GREENLIGHT in educational contexts to instil energy-aware coding habits in classrooms.

7 ACKNOWLEDGEMENT

Saurabhsingh Rajput and Tushar Sharma are supported through grant NSERC Discovery RGPIN/04903. Maria Kechagia and Federica Sarro are supported by the European Research Council under grant no. 741278 (EPIC).

REFERENCES

[1] Dario Amodei and Danny Hernandez. 2018. AI and Compute. https://openai.com/blog/ai-and-compute/

[2] Code Carbon. 2023. Code Carbon. https://github.com/mlco2/codecarbon

[3] Junyi Chai, Hao Zeng, Anming Li, and Eric WT Ngai. 2021. Deep learning in computer vision: A critical review of emerging techniques and application scenarios. *Machine Learning with Applications* 6 (2021), 100134.

[4] Yunji Chen, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. 2016. DianNao family: energy-efficient hardware accelerators for machine learning. *Commun. ACM* 59, 11 (2016), 105–112.

[5] Shi Dong, Ping Wang, and Khushnood Abbas. 2021. A survey on deep learning and its applications. *Computer Science Review* 40 (2021), 100379.

[6] Stefanos Georgiou, Maria Kechagia, Tushar Sharma, Federica Sarro, and Ying Zou. 2022. Green AI: Do Deep Learning Frameworks Have Different Costs?. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 1082–1094. https://doi.org/10.1145/3510003.3510221

[7] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. 2021. Knowledge distillation: A survey. *International Journal of Computer Vision* 129 (2021), 1789–1819.

[8] Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. 2020. Towards the Systematic Reporting of the Energy and Carbon Footprints of Machine Learning. arXiv:2002.05651 [cs.CY]

[9] Monsoon Solutions Inc. [n. d.]. High Voltage Power Monitor. https://www.msoon.com/online-store/High-Voltage-Power-Monitor-p90002590

[10] Zhihao Jia, Oded Padon, James Thomas, Todd Warszawski, Matei Zaharia, and Alex Aiken. 2019. TASO: optimizing deep learning computation with automatic generation of graph substitutions. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 47–62.

[11] Dario Lazzaro, Antonio Emanuele Cinà, Maura Pintor, Ambra Demontis, Battista Biggio, Fabio Roli, and Marcello Pelillo. 2023. Minimizing Energy Consumption of Deep Learning Models by Energy-Aware Training. In *International Conference on Image Analysis and Processing*. Springer, 515–526.

[12] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. 2021. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing* 461 (2021), 370–403.

[13] Alexandra Sasha Luccioni, Yacine Jernite, and Emma Strubell. 2023. Power Hungry Processing: Watts Driving the Cost of AI Deployment? arXiv:2311.16863 [cs.LG]

[14] Linux manual page. 2023. perf-stat— Linux manual page. https://man7.org/linux/man-pages/man1/perf-stat.1.html.

[15] Zhi Qi, Weijian Chen, Rizwan Ali Naqvi, and Kamran Siddique. 2022. Designing Deep Learning Hardware Accelerator and Efficiency Evaluation. *Computational Intelligence and Neuroscience* 2022 (2022).

[16] Saurabhsingh Rajput, Tim Widmayer, Ziyuan Shang, Maria Kechagia, Federica Sarro, and Tushar Sharma. 2023. FECoM: A Step towards Fine-Grained Energy Measurement for Deep Learning. *arXiv preprint arXiv:2308.12264* (2023).

[17] Rajput Saurabhsingh. 2023. CodeGreen. https://github.com/SMART-Dal/codegreen

[18] Rajput Saurabhsingh. 2023. Greenlight. https://github.com/SMART-Dal/greenlight/tree/main/analysis/cumulative_data

[19] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. 2020. Green AI. *Commun. ACM* 63, 12 (Nov. 2020), 54–63. https://doi.org/10.1145/3381831

[20] Arjan van de Ven. [n. d.]. The Linux PowerTOP Tool. https://github.com/fenrus75/powertop