Influence of Hyperparameters and Hardware Configurations on Energy Consumption in Fine-Tuning an NLP Model

Author

Denergium - More FLOPS per Watt!

Contact : contact@denergium.com

Abstract

This study investigates the impact of hyperparameters (batch size, numerical precision) and hardware configurations (GPU type, power capping) on energy efficiency and performance during the fine-tuning of an NLP model. Using EnergyScopium, a specialized software suite developed by Denergium for precise energy profiling of HPC and AI workloads, we measured CPU/GPU energy consumption, efficiency metrics, and runtime across multiple infrastructures. Key findings reveal that strategic hyperparameter tuning (e.g., batch size 64 with fp16 precision) achieves 94.3% accuracy while consuming 470 Wh on Meluxina, whereas smaller batches (8–32) increase runtime by 30% with marginal energy gains. Notably, power capping shows a reduced energy consumption by 10% with only a 6% runtime increase on Grid'5000, demonstrating a favorable efficiency-runtime trade-off . Clustering analysis not clearly identifies distinct training profiles, emphasizing the complexity of software-hardware co-optimization. Future work will extend this methodology to broader HPC-AI applications, integrating triadic constraints (accuracy, energy, duration) to advance sustainable computing practices.

Introduction

Deep learning models have revolutionized many fields, but their training is known for being computationally intensive and energy-demanding. As models grow in size and complexity, the energy required for training also increases significantly. Therefore, optimizing the use of computational resources and minimizing energy consumption is critical for both environmental sustainability and cost-effectiveness. Hyperparameters play a crucial role in determining the efficiency of model training.

This study aims to explore how various hyperparameters influence the energy consumption of deep learning models, specifically during the fine-tuning of the DistilBERT model. By focusing on a range of parameters, we aim to provide a more comprehensive understanding of how these factors interact and their combined effect on both model performance and energy efficiency. To achieve this, we use **EnergyScopium**, a tool that allows us to precisely measure energy usage across different components of the training pipeline, including CPU and GPU.

Our objective is to incorporate energy consumption as a key metric alongside model quality and fine-tuning duration. By doing so, we extend traditional evaluations that primarily focus on accuracy and training time to include an energy-aware perspective. Through this investigation, we aim to identify the most efficient configurations in terms of energy consumption, without sacrificing model accuracy, and provide actionable insights for optimizing the energy footprint of deep learning tasks.

Experimental Setup

In this section, we present the computational resources used for our experiments. Our setup includes a diverse set of high-performance computing (HPC) clusters, each providing different hardware configurations and capabilities suited for our workload.

We had access to Cali3, a computing cluster attached to the Mésocentre de Calcul Intensif Aquitain (MCIA), offering a local infrastructure for initial testing and benchmarking. Additionally, we leveraged Meluxina, a supercomputer operated by LuxProvide in Luxembourg, known for its high computational power and scalability. To extend our experiments across a distributed environment, we utilized Grid'5000, a large-scale and flexible testbed for research in distributed and parallel computing.

The combination of these resources allowed us to conduct a comprehensive evaluation, benefiting from different architectures, scheduling policies, and execution environments. The following subsections provide further details on the specific hardware configurations of each machine.

Cali3

Hardware configuration

Component	Specifications
Processors	2x AMD EPYC 9254 (24 cores each)
RAM	384 GB

Accelerators	4x Nvidia L40

Environment Software Configuration

Software	Version
Operating System	AlmaLinux 9.40
Python	3.10.6
PyTorch	2.60.0
CUDA	12

SLURM Configuration

Parameter	Value
Nodes	1
GPU	4
Memory	192G
Number of tasks	1
CPUs per task	48

Meluxina

Hardware Configuration

Component	Specifications
Processors	2x AMD EPYC 7452 (32 cores each)
RAM	512 GB
Accelerators	4x Nvidia A100 40GB

Environment Software Configuration

Software	Version
Operating System	Red Hat Enterprise Linux 8
Python	3.10.6
PyTorch	2.60.0
CUDA	12

SLURM Configuration

Parameter	Value
Nodes	1
GPU	4
Memory	256G
Number of tasks	1
CPUs per task	64

GRID5000

Hardware Configuration

Component	Specifications
Processors	2x AMD EPYC 7452 (32 cores each)
RAM	512 GB
Accelerators	4x Nvidia A100 40GB

Environment Software Configuration

Software	Version
Operating System	Red Hat Enterprise Linux 8
Python	3.10.6
PyTorch	2.60.0
CUDA	12

OAR Configuration

Parameter	Value
Nodes	1
GPU	4
Memory	256G
Number of tasks	1
CPUs per task	64

Environment Software Configuration

Software	Version
Operating System	Red Hat Enterprise Linux 8
Python	3.10.6
PyTorch	2.60.0
CUDA	12

SLURM Configuration

Parameter	Value
Nodes	1
GPU	1
Memory	256G
Number of tasks	1
CPUs per task	72

Hardware Overview and GPU Architecture Timeline

The experimental evaluation was conducted on four different high-performance computing platforms, each featuring distinct generations of NVIDIA GPUs. This diversity allows for a meaningful analysis of how GPU architecture evolution impacts energy consumption and computational efficiency in fine-tuning scenarios.

The platforms span three major NVIDIA architectures: - **Ampere (A100)**: Released in 2020, Ampere brought significant improvements in compute throughput, tensor core support, and memory bandwidth. It is widely regarded as the workhorse for large-scale training workloads. - **Ada Lovelace (L40)**: Introduced in late 2022, this architecture primarily targets graphics and AI inference, though it offers respectable compute capabilities with FP8 and sparsity support.

Application: Model and Hyperparameters

Hugging Face has emerged as a leading platform for natural language processing (NLP) and machine learning, offering a vast collection of pre-trained models, efficient inference tools, and user-friendly APIs. For our application, we leverage Hugging Face's ecosystem to streamline model fine-tuning, benefiting from its extensive library of state-of-the-art transformer models.

By utilizing Hugging Face, we gain access to optimized implementations of deep learning architectures, allowing us to efficiently experiment with well known models while minimizing development overhead.

The following reference (https://huggingface.co/spaces/nanotron/ultrascale-playbook) has given us a great deal of insight into the influence of hyperparameters.

In this subsection, we detail how Hugging Face is incorporated into our application, covering aspects such as model selection and training strategies.

Parameter	Value
Model	DistilBERT
Dataset	AG News
Optimizer	AdamW (torch fused)
Learning Rate	5e-5
Number of Epochs	5
Loss Function	CrossEntropyLoss
Metrics	Accuracy, Loss

Several forms of parallelism can be employed during training to accelerate computation and handle large-scale models. Data parallelism replicates the model across multiple devices and splits the input data, enabling synchronous gradient updates after each batch. Tensor parallelism divides individual layers (especially matrix multiplications) across devices, often used in very large models. Pipeline parallelism breaks the model into sequential stages, each assigned to different devices, allowing overlapping of forward and backward passes for improved throughput. Context parallelism, less common, distributes attention heads or sequence tokens across devices, useful in handling long-context tasks. In our study, we chose data parallelism due to its simplicity and effectiveness in fine-tuning transformer models on multi-GPU setups. This allowed us to reduce training time while maintaining consistency in model performance.

Although we have included quality as a parameter, for the purposes of this study we only consider values of sufficient quality to assess the fine-tuning as a success.

Quality was evaluated using Accuracy and Loss metrics, with our quality parameter choices spanning the observed minimum and maximum values. Starting from a pre-trained model (which establishes our baseline quality), we proceeded with fine tuning—incorporating brief press releases in our case—to further improve performance. Based on discussions in various forums, a range of 3 to 5 epochs was considered suitable for fine tuning, and we ultimately chose to run for **5 epochs**.

The training were expected to last an approximate duration of 1500 seconds and an energy consumption of about 500 Wh.

Energy Consumption Measurement: The EnergyScopium software suite

Denergium (www.denergium.com) is an Inria startup, founded in 2023. Denergium improves the efficiency of supercomputing. Our customers are major players in industry, academia, government agencies, BFSI and Pharma, for whom simulation and AI are major challenges. Thanks to our innovation, based on an understanding of the energy involved in computations, we can provide software solutions and expertise to improve energy management and, ultimately, increase the number of computations.

EnergyScopium is a software suite developed by Denergium. EnergyScopium is available in 2 versions: EnergyScopium Smart Decision and EnergyScopium profiler.

EnergyScopium Smart Decision gives HPC infrastructure managers qualitative feedback on the energy efficiency of all jobs and servers. As a result, they can improve infrastructure efficiency, and ultimately perform more computations.

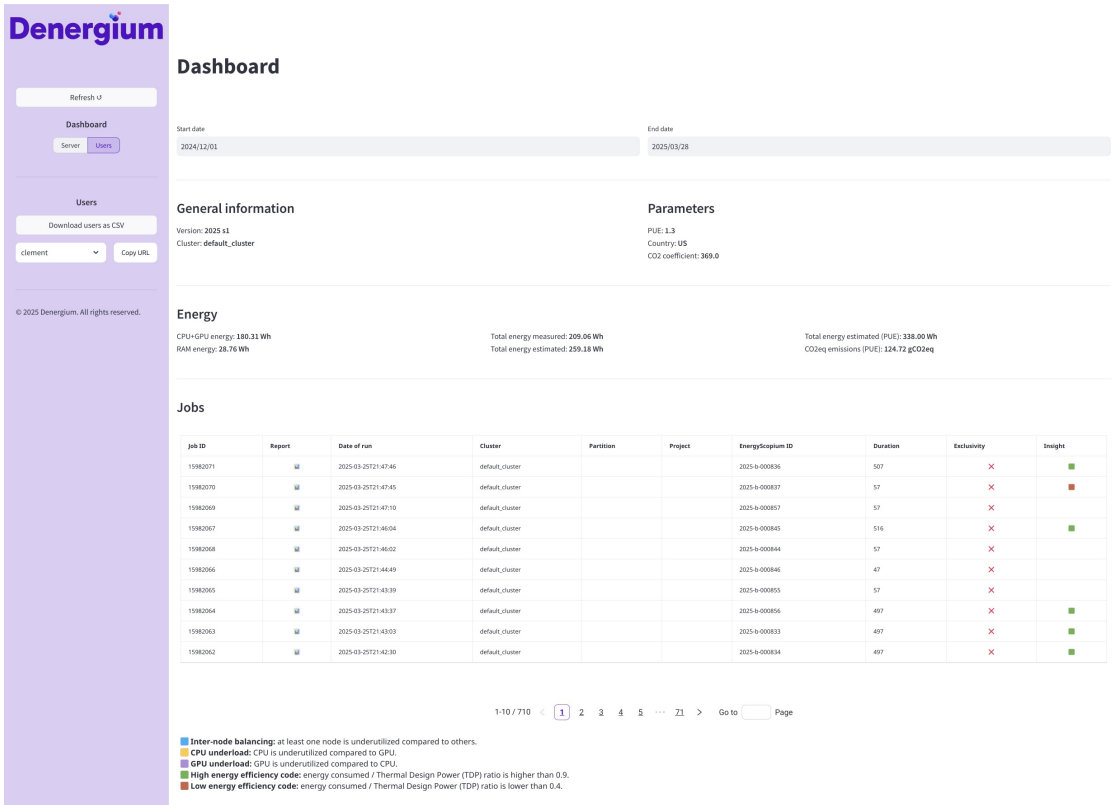
EnergyScopium also provides users and developers with different levels of information:

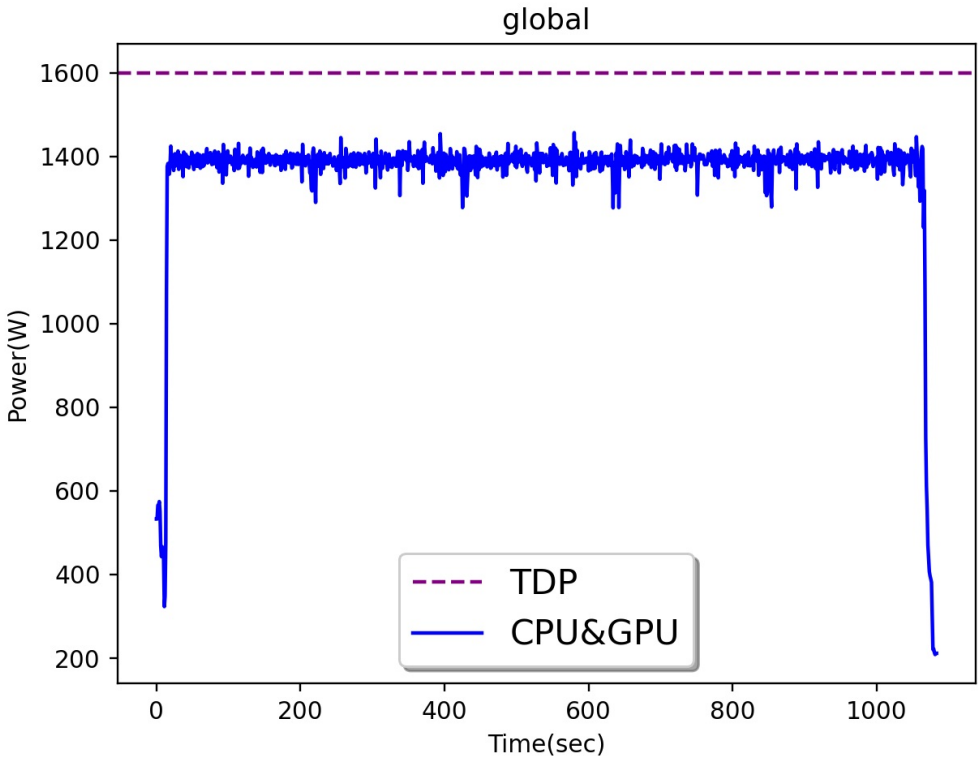
- an efficiency label
- energy profile data
- insights into efficiency quality

EnergyScopium profiler is dedicated to users and developers who want to compare, improve and measure their applications from an energy angle. We use EnergyScopium profiler to collect the data in this study.

EnergyScopium provides information on the entire energy chain

Energyscopium precisely measures the energy consumed by each physical unit and calculates the efficiency—such as the ratio of CPU energy usage to its Thermal Design Power (TDP). These raw measurements are then refined through estimations that integrate the overall node consumption. The PUE (Power Usage Effectiveness) is then taken into account to calculate consumption as close as possible to reality. Thanks to our knowledge of the energy mix of the place where the calculation was made, we can propose a consumption equivalent in gCO2 for the calculation.





First figure shows the Denergium dashboard, part of the EnergyScopium suite, summarizing job energy metrics, where most jobs exhibit low exclusivity and suboptimal resource usage, flagged by color-coded inefficiency insights.

Second figure displays a global power profile where the system maintains stable consumption (~1400 W) well below the TDP (1600 W), indicating consistent load but potential underutilization of peak capacity. Note that the blue curve indicates the power dissipated (Watt), and the area under the blue curve indicates the energy consumed (Joule).

Experimental Protocol

In machine learning, hyperparameters are predefined configurations that govern the training process without being updated by the learning algorithm itself. They include values such as batch size, learning rate, or numerical precision, and can strongly influence model performance, convergence time, and resource consumption.

More broadly, we refer to all adjustable variables in our study as experimental parameters, which include both traditional hyperparameters and hardware-specific configurations. While hyperparameters such as batch size, number of data loader workers, or gradient accumulation steps are central to the training logic, hardware-level parameters—such as the choice of GPU, power capping thresholds, or clock frequency settings—also play a critical role in shaping energy consumption and runtime performance. In this work, both types of parameters were systematically varied to evaluate their combined impact on the energy efficiency and performance of fine-tuning DistilBERT on the AG News dataset.

Hardware Combinatorics

We evaluated energy efficiency and performance across three distinct NVIDIA GPU architectures: the NVIDIA A100, NVIDIA A100 SXM, and NVIDIA L40. These architectures offer diverse profiles in terms of raw computational power, memory bandwidth, and thermal design, allowing us to analyze trade-offs between energy use and training speed.

For the A100 and L40, standard operating conditions were maintained throughout the experiments to establish performance and energy baselines. The A100 SXM platform, hosted on Grid5000, enabled us to go further by experimenting with power capping and clock frequency scaling:

- Power caps: 100% (regular mode), 85%, 70%, 50% (with respect to the GPU TDP)
- GPU Clock frequencies: 100% (regular mode), 85%, 70%

Importantly, power capping and clock frequency were varied independently (i.e., when one varied, the other

was kept at 100%).

These hardware-level tuning experiments can help demonstrating that beyond algorithmic optimizations, low-level hardware configuration can significantly impact energy efficiency and runtime, offering additional levers to reduce the cost and environmental footprint of deep learning workloads.

Hyperparameters Combinatorics

Beyond hardware variations, we systematically explored how different training hyperparameters influence energy consumption and model performance. This combinatorial study includes parameters such as batch size, numerical precision, number of data loader workers, gradient accumulation steps, and TPU cores usage. Tensor Processing Units (TPUs) cores were included in select configurations to assess their impact compared to traditional GPU cores training. Each configuration was carefully selected to capture a broad yet relevant search space across different platforms.

Cali3

Hyperparameter	Tested Values
Train Batch size	[8, 16, 32, 64]
Eval Batch size	[8, 16, 32, 64]
Numerical precision	[fp32, fp16]
Number of workers	[1, 2, 4, 8, 16]
Number of TPU cores	[0, 512]
Gradient accumulation steps	[1, 2, 4, 8, 16]

Combinatorics : 4 x 2 x 5 x 2 x 5 = 400

Note: **Eval batch size** and **train batch size** are always linked together and kept the same in all experiments.

Meluxina

Hyperparameter	Tested Values
Train Batch size	[8, 16, 32, 64]
Eval Batch size	[8, 16, 32, 64]
Numerical precision	[fp16]
Number of workers	[4, 8, 32, 64]
Number of TPU cores	[0, 432]
Gradient accumulation steps	[1, 2, 4, 8]

Combinatorics : 4 x 1 x 4 x 2 x 4 = 128

Note: **Eval batch size** and **train batch size** are always linked together and kept the same in all experiments.

Grid 5000

Hyperparameter	Tested Values
Train Batch size	[64]
Eval Batch size	[64]
Numerical precision	[fp16]
Number of workers	[24]
Number of TPU cores	[0]
Gradient accumulation steps	[4]

Hardware parameter	Tested Values
Power Capping	[50%, 70%, 85%, 100%]
Frequency	[70%, 85%, 100%]

Combinatorics : 6

Here we decided to only run the most efficient hyperparameters configuration thanks to our previous runs on Cali3 and Meluxina.

For each combination of our experimental parameters, we measured the energy consumption using **EnergyScopium**, which provided the following metrics:

- Total energy consumption
- CPU energy efficiency
- GPU energy efficiency
- Global energy efficiency
- The full energy trace

Clustering

Scikit-learn is a widely used machine learning library in Python, offering a comprehensive set of tools for data preprocessing, supervised and unsupervised learning, model evaluation, and dimensionality reduction. Its

efficient and user-friendly API makes it ideal for rapid experimentation and integration into various workflows.

For our application, we leverage scikit-learn to perform key tasks such as data standardization, clustering, and dimensionality reduction. The library's implementations of K-Means, Principal Component Analysis (PCA), and other essential techniques allow us to efficiently analyze and structure our data. In the following subsections, we detail how scikit-learn is utilized in our pipeline, from preprocessing raw data to clustering and visualizing results.

Data aggregation

Each combination of experimental parameters, spanning both hyperparameters and hardware configurations, produced a corresponding set of performance and energy-related measures. These include metrics such as runtime, accuracy, and various energy efficiency indicators. Together, the experimental parameters and resulting measurements constitute the full set of variables in our study. These variables serve as the foundation for our downstream analysis: we represent each experimental run as a vector in this multi-dimensional space, which will be used for clustering and pattern discovery across the different configurations.

Variable	Description	Tunability
Batch size	Batch size used	Tunable
Numerical Precision	fp32 or fp16	Tunable
Gradient Accumulation Steps	Number of gradient accumulation steps	Tunable
Number of Workers	Number of workers for the DataLoader	Tunable
Number of TPU Cores	Number of TPU cores used	Tunable
Power Capping	Hardware parameter	Tunable
Frequency	Hardware parameter	Tunable
CPU Efficiency	CPU energy efficiency	Non Tunable
GPU Efficiency	GPU energy efficiency	Non Tunable
Global Efficiency	Overall energy efficiency	Non Tunable
Runtime	Total execution time	Non Tunable
Accuracy	Model accuracy	Non Tunable
Evaluation Loss	Validation loss	Non Tunable
Energy Consumed	Total energy consumption	Non Tunable

Strategies

We explored different clustering strategies to analyze our dataset effectively, each with its own advantages and trade-offs:

- **Result-Based Clustering**:
  - This method groups executions based solely on their outcomes, such as energy consumption, runtime, or accuracy.
  - Once clusters are formed, we analyze the associated variables to identify trends and optimal configurations.
  - This approach is particularly useful for understanding the structure of results before investigating parameter influence.
- **Parameter-Result Clustering**:
  - This approach clusters executions based on both input parameters and output results, forming groups with similar configurations and behaviors.
  - By incorporating tunable hyperparameters into the clustering process, we gain deeper insights into how specific settings influence performance and efficiency.
  - However, careful preprocessing is required to ensure balanced feature importance and prevent certain variables from dominating the clustering process.
- **Hybrid Approach**:
  - A two-step process where we first cluster based on results to identify performance patterns, then analyze the distribution of input parameters within each group.
  - If necessary, a second clustering step incorporating all the variables refines the segmentation, leading to more actionable insights.

Each of these strategies plays a role in structuring our dataset, allowing us to extract meaningful patterns in energy efficiency, runtime, and performance trade-offs. Our overall approach is hybrid, with an initial focus on parameters to structure the execution space, followed by result-based analysis to refine interpretation. This allows us to identify broad families of configurations before associating them with energy efficiency or performance profiles.

Data processing

A clustering process using scikit-learn typically involves three main steps: data preprocessing, clustering, and visualization.

- Data Preprocessing:
  - Load the dataset and handle missing values if necessary.
  - Standardize the features using StandardScaler to ensure all variables contribute equally to the clustering process.
- Clustering with K-Means:
  - Apply K-Means from sklearn.cluster, specifying the number of clusters (k).

- Fit the model to the data and retrieve the cluster labels.
- Visualization with PCA (Principal Component Analysis):
  - Reduce the dataset's dimensionality to 2D or 3D using Principal Component Analysis (PCA) from sklearn.decomposition.
  - Plot the data points, coloring them according to their assigned clusters.

Data Visualizations

We developed a **Streamlit** interface to visualize our results and make our clustering process interactive. This interface allows users to explore the data dynamically, and observe how different clustering parameters affect the dataset. To enhance visualization, we integrated **Plotly** for generating interactive and responsive plots. By leveraging Streamlit's and Plotly's capabilities, we provide an intuitive and user-friendly way to interact with the clustering results, facilitating better analysis and interpretation.

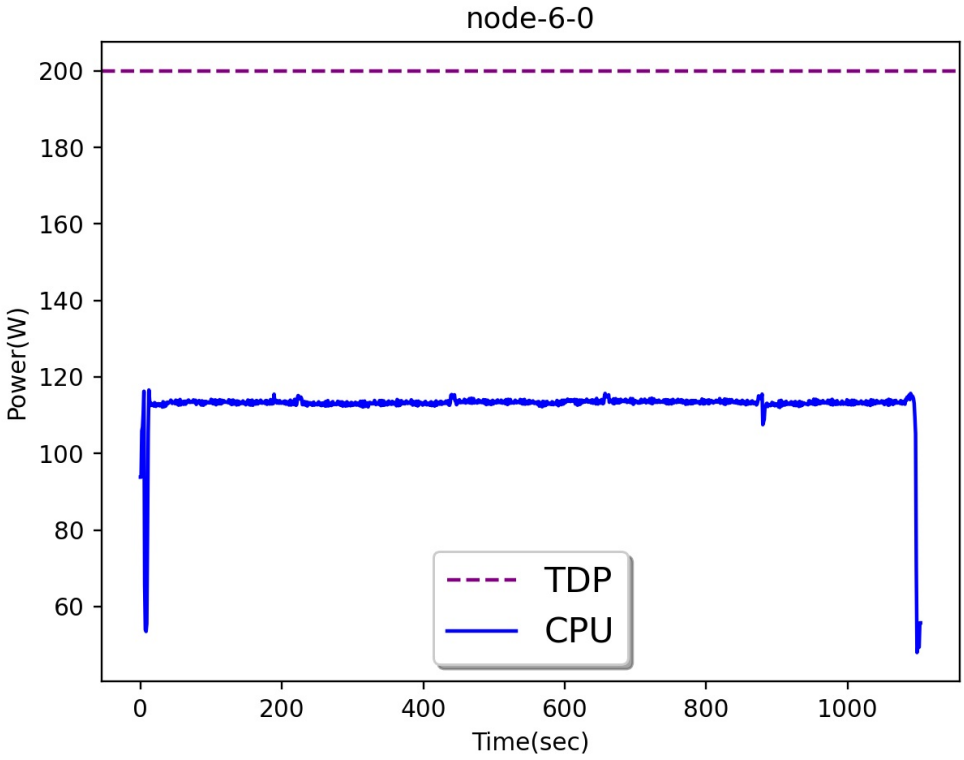
The image below illustrates a screenshot of our **Streamlit** interface, designed for interactive clustering. This tool enables users to explore and refine cluster assignments dynamically, facilitating an intuitive and efficient analysis of complex datasets.

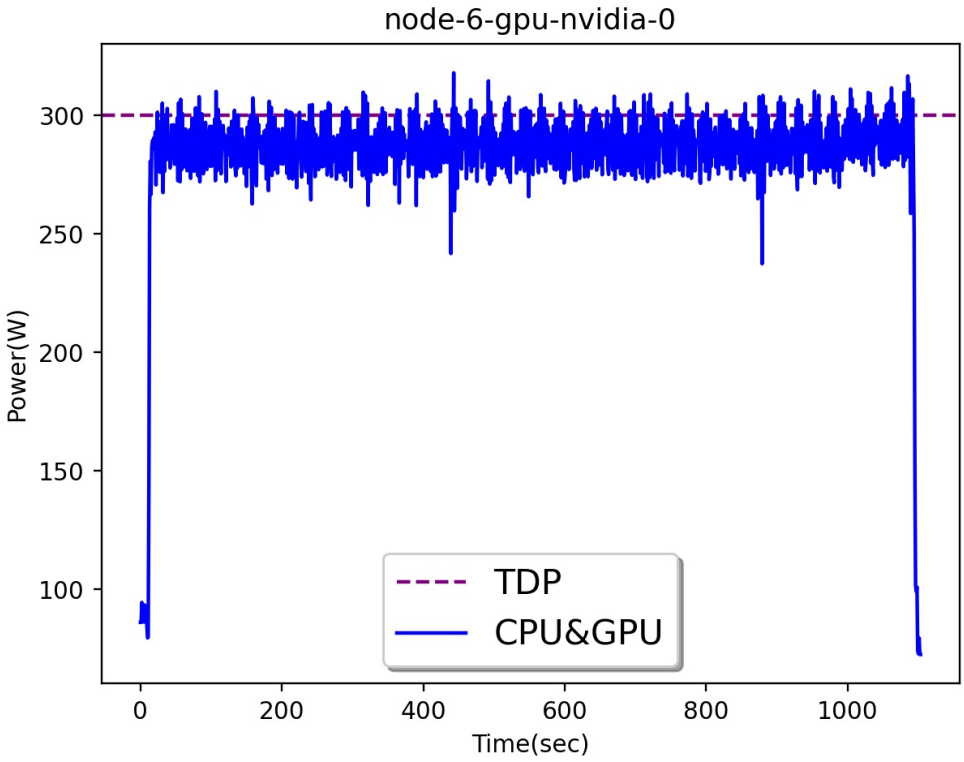
Results

In this section, we present the results from our comprehensive analysis of over 500 fine-tuning jobs—totaling approximately 250 kWh of energy consumption—conducted across various machines. Our study investigates the intricate relationship between runtime, energy usage, and model accuracy using a custom energy metric. We begin by comparing different energy profiles to emphasize how efficiency varies across GPU configurations and hyperparameter settings. Next, we leverage clustering techniques to identify groups of hyperparameter configurations that effectively balance performance with energy consumption. Finally, we explore the strong correlation between runtime and total energy usage, highlighting the significant influence of batch size and other hyperparameters on overall efficiency.

GPU Computation in Fine-Tuning DistilBERT

Fine-tuning DistilBERT is predominantly a GPU-driven process due to the intensive matrix multiplications and attention mechanisms inherent in transformer architectures. During fine tuning, each batch undergoes forward and backward propagation—operations that rely heavily on parallelizable tensor computations. GPUs, with their thousands of cores and specialized hardware (such as tensor cores), are exceptionally well-suited for these tasks, enabling them to execute floating-point operations much faster than conventional CPUs.





These two plots clearly illustrate a GPU-centric workload:

- In the CPU power profile (first plot), the consumption remains stable around 115W, well below its 200W

TDP, indicating modest CPU usage.

- In contrast, the GPU power profile (second plot) shows sustained, near-maximum consumption around 300W (its TDP), with a dense and active power signature, confirming that the workload is compute-intensive on the GPU.

Correlation between energy and runtime

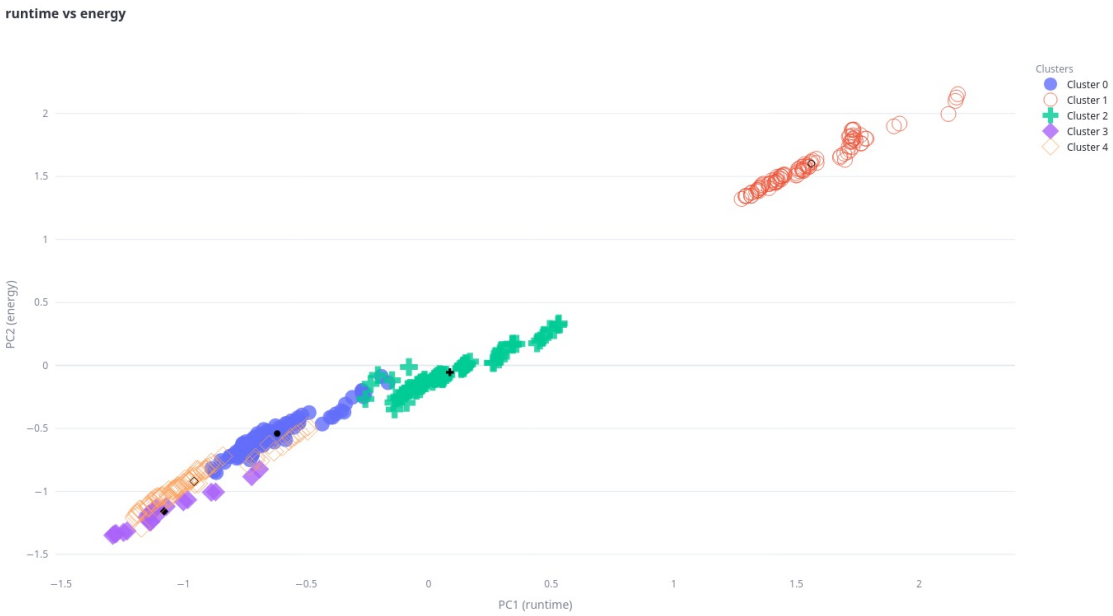


Figure below illustrates the 2D projection of the energy-runtime space. We observe that the energy consumption and execution time are strongly correlated, with each GPU architecture showing a distinct slope in this space.

The slope, which corresponds to the energy-to-runtime ratio, remains constant within a given architecture (e.g., A100, L40), despite variations in frequency and power capping. This suggests that GPU clock frequency tuning has little influence on this metric in our case. Unfortunately, we could not experimentally vary GPU memory frequency, which might have offered additional insight.

Interestingly, the difference in slopes between architectures highlights a hardware-specific energy efficiency. We quantify this efficiency using the following metric:

```
Efficiency = Energy consumed / (TDP × Runtime)
```

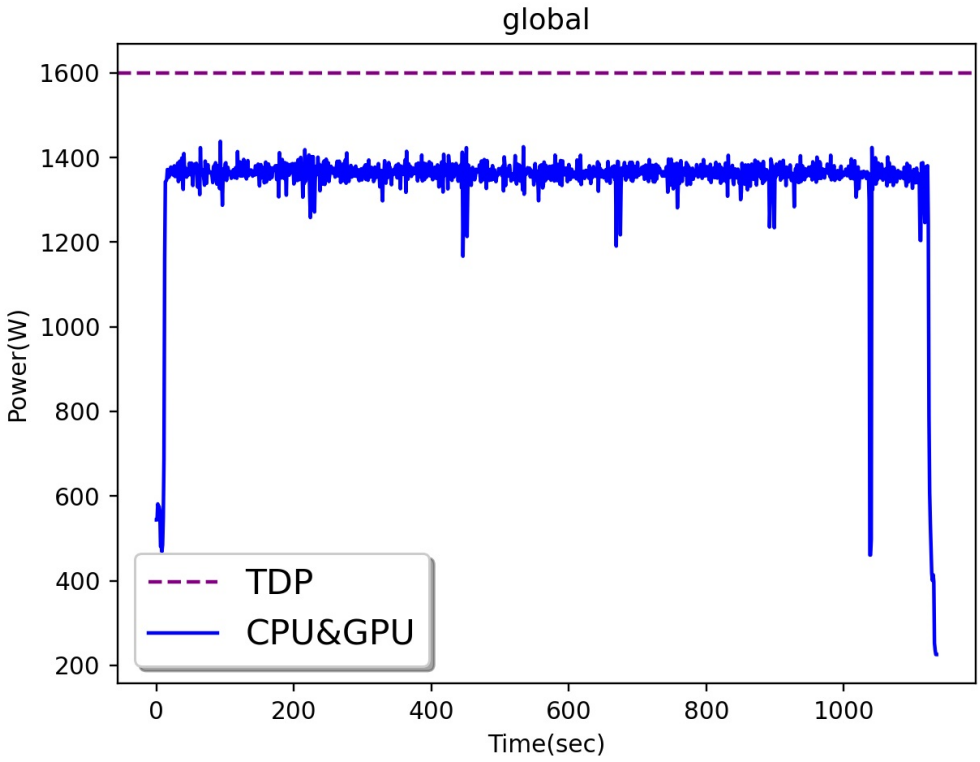
This metric and the observed PCA (Principal Component Analysis) plot both confirm the relevance of our efficiency definition.

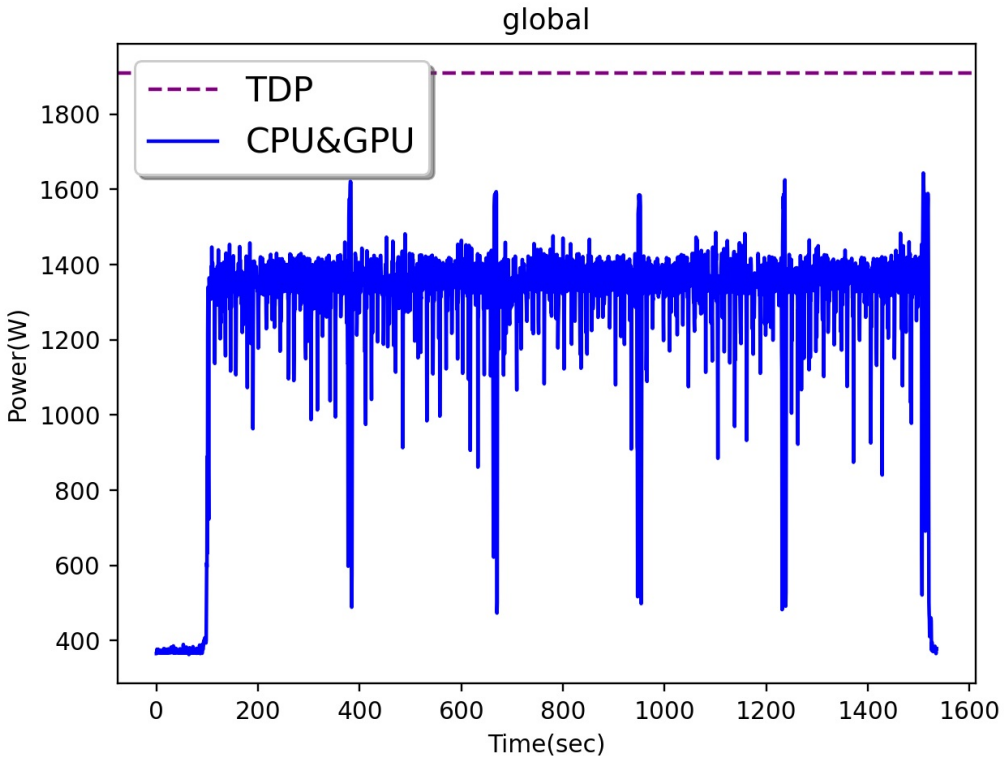
Energy Efficiency Comparison Between L40 and A100 GPUs

Our analysis of the energy profiles for the A100 and L40 GPUs reveals a clear difference in energy efficiency. While both GPUs achieve equivalent fine-tuning results, the A100 exhibits a significantly larger gap between its power consumption curve and its maximum thermal design power (TDP) compared to the L40. This indicates that the A100 operates at a lower efficiency relative to its theoretical power capacity.

The more recent L40 seems to be more efficient for theses runs, considering the study environment.

This reduced efficiency translates into lower effective computational throughput, leading to a longer fine-tuning duration on the A100. As a result, even though both GPUs ultimately consume similar amounts of power at any given moment, the extended training time on the A100 leads to a **higher total energy consumption**.





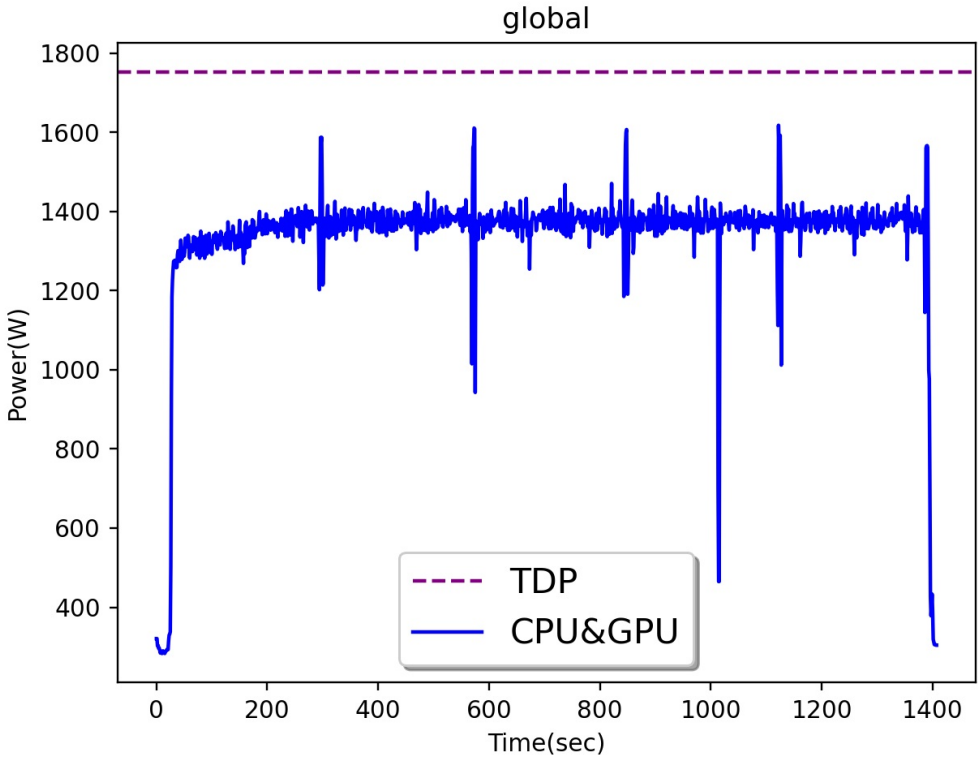
First figure shows the profile with a L40 GPU, the power consumption (~1400 Watt) is close to the TDP (1600 Watt). Second figure shows the profile with a A100 GPU, the power consumption (~1400 Watt) is not so close to the TDP (1900 Watt)..

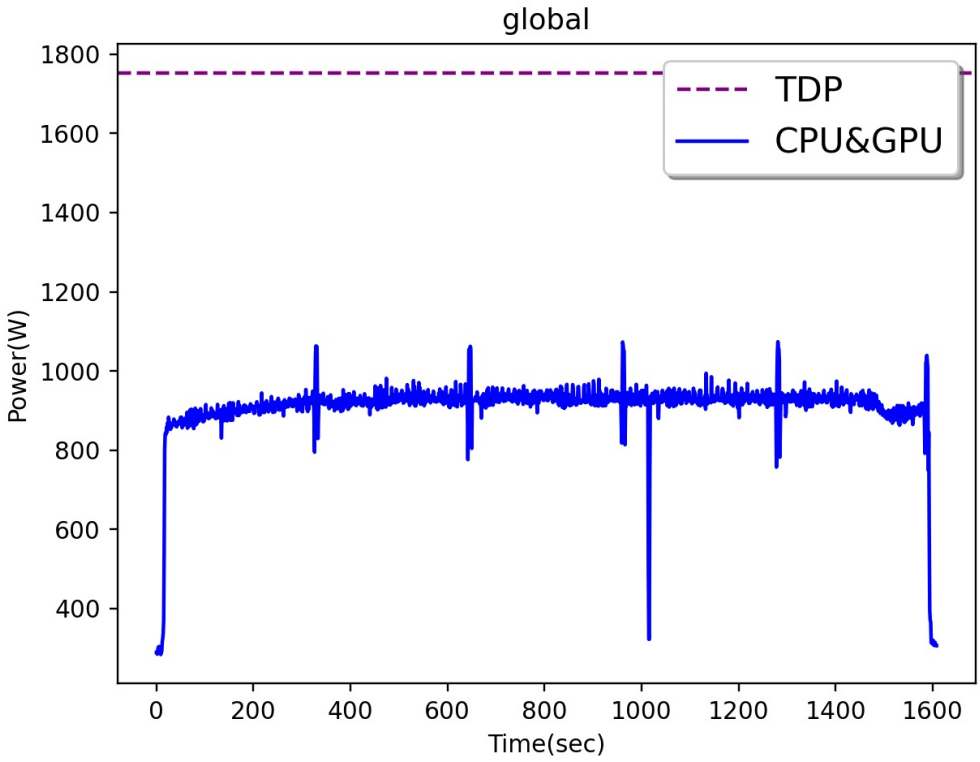
Power Capping and Clock Frequencies

In this section, we explore two strategies for reducing energy consumption during job execution: modifying clock frequencies and applying power capping. Each approach is compared against a nominal case, where the job runs without any limitations on frequency or power.

Clock Frequencies Scaling

We first experimented with lowering the clock frequency, effectively reducing the power draw at the cost of longer runtimes. The three plots below illustrate the effect of frequency scaling on power and performance:





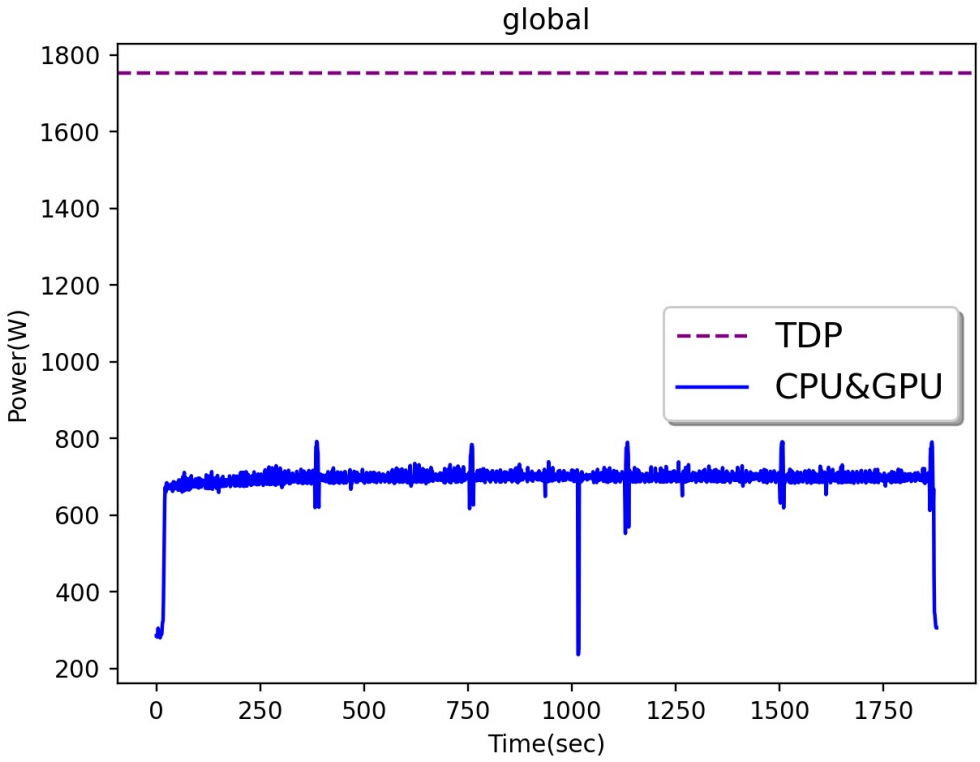


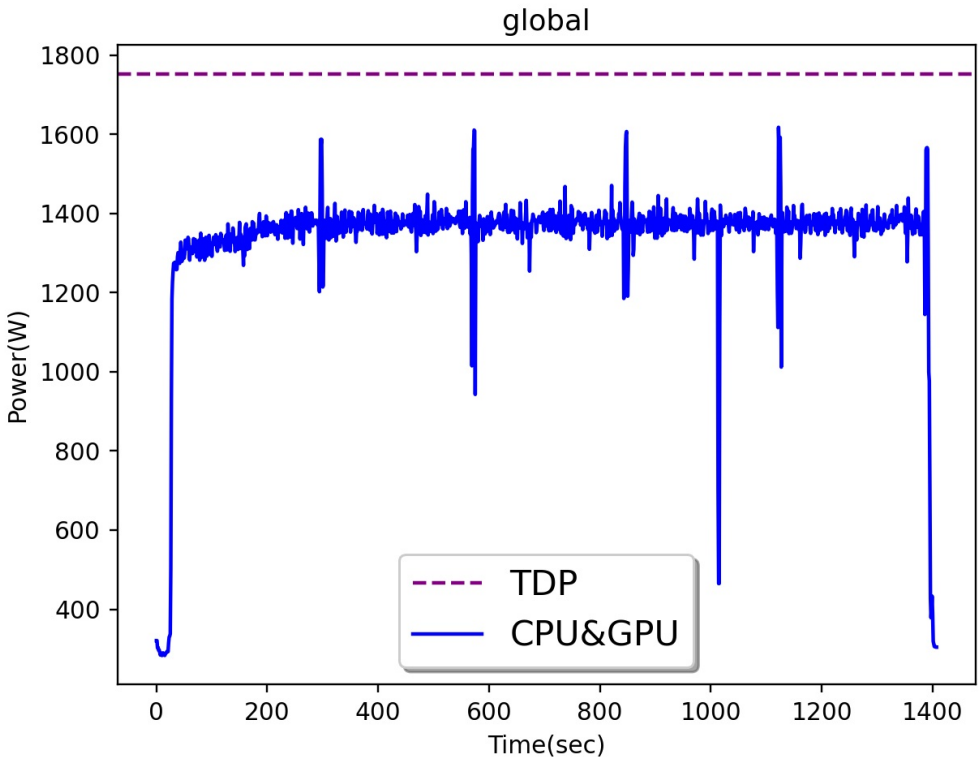
Figure (freq)	Duration (s)	Power (W)	Energy (Wh = W x sec / 3600)
1 (100%)	1400 (ref)	1400 (ref)	544 (ref)
2 (85%)	1600 (+14%)	950 (-33%)	420 (-23%)

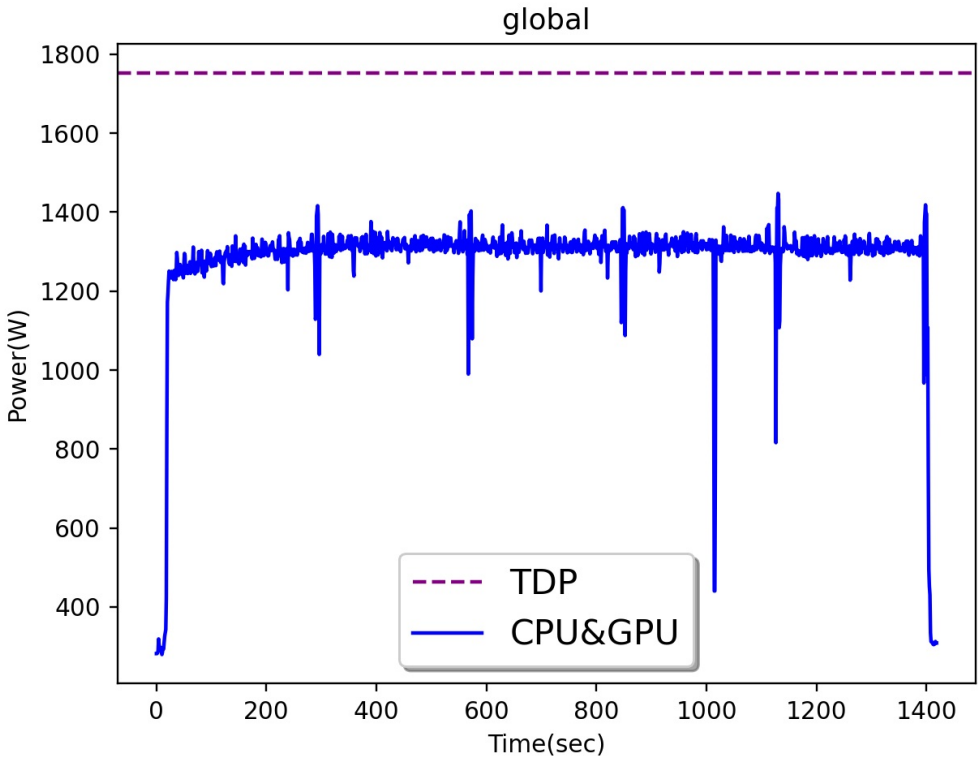
3 (70%)	1850 (+32%)	700 (-50%)	360 (-34%)

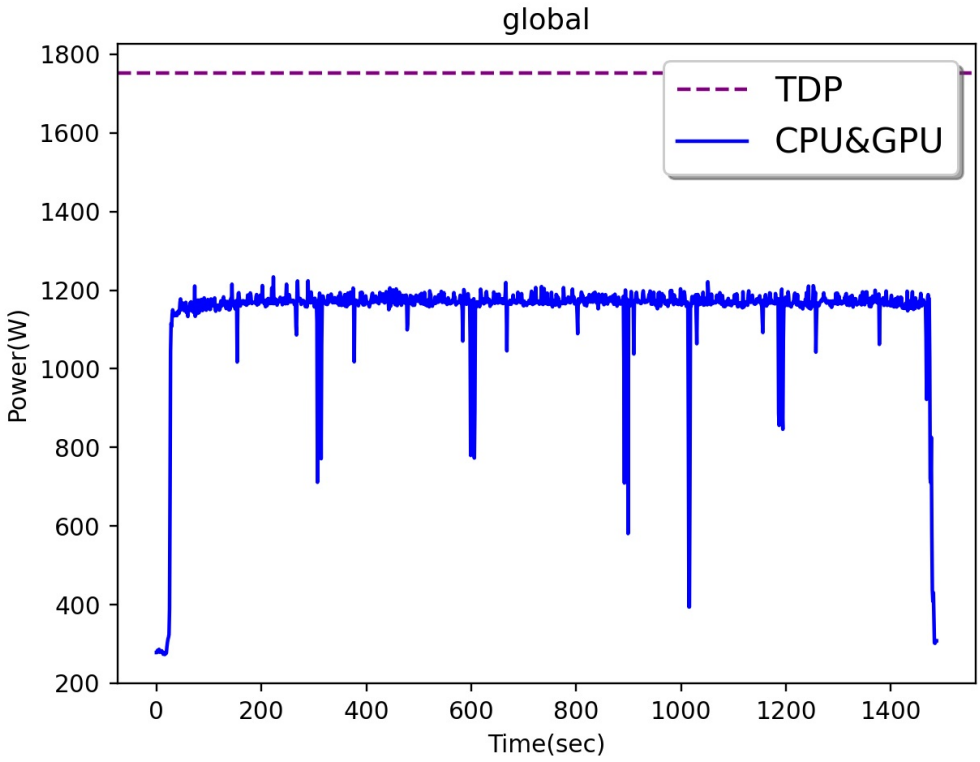
- A 14% runtime increase at 85% frequency reduces power by 33% and energy by 23%.
- At 70%, runtime slows 32%, but power drops 50%, yielding 34% energy savings.
- Clock frequencies scaling thus provides a predictable trade-off, enabling operators to select frequency levels that match their energy-delay budgets.

Power Capping

Next, we apply power caps, which limit the maximum allowable power usage of the job without explicitly modifying the clock frequency. This method allows hardware to self-adjust while staying within the specified budget.




global
1800
1600
1400
1200
1000
800
600
400
Power(W)
TDP
CPU&GPU
0
200
400
600
800
1000
1200
1400
Time(sec)


global
1800
1600
1400
1200
1000
800
600
400
200
Power(W)
TDP
CPU&GPU
0
200
400
600
800
1000
1200
1400
Time(sec)

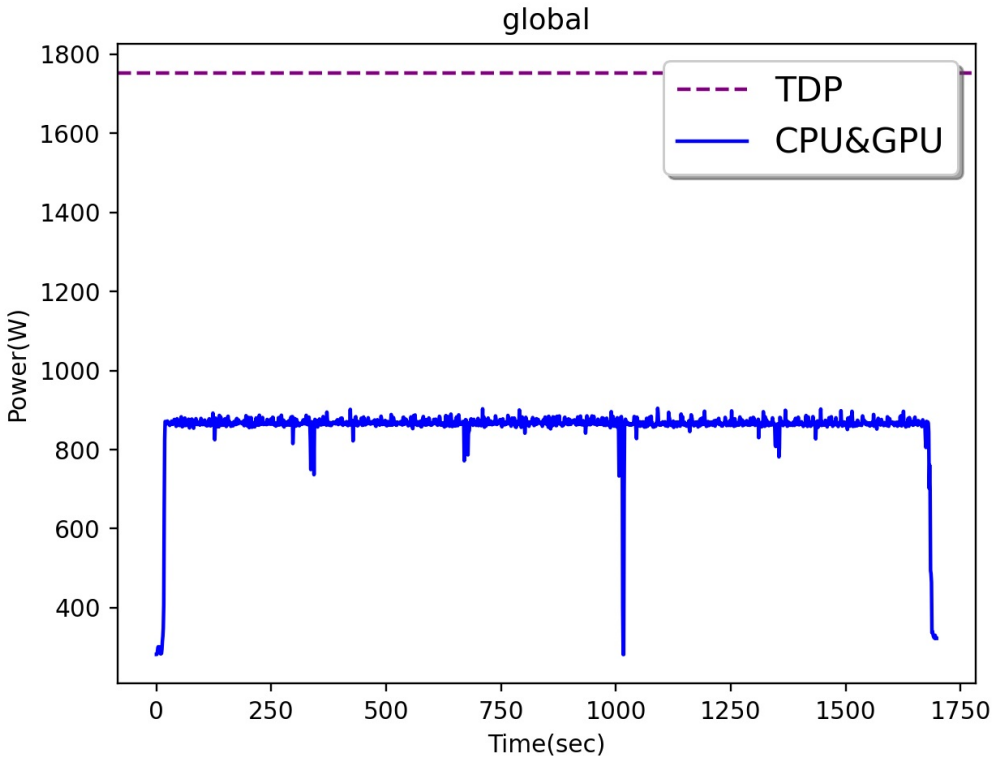


Figure (PowerCapping %TDP)	Duration (s)	Power (W)	Energy (Wh = W x sec / 3600)
1 (100%)	1400 (ref)	1400 (ref)	544 (ref)
2 (85%)	1400 (+0%)	1350 (-4%)	525 (-4%)
3 (70%)	1500 (+7%)	1200 (-15%)	500 (-8%)
4 (50%)	1700 (+32%)	850 (-39%)	401 (-26%)

- Capping at 85 % incurs no runtime increase yet saves 4 % energy, an efficient first step.
- At 70 %, a 7 % slowdown yields 15 % power reduction and 8 % energy savings.
- More aggressive caps (50 %) slow jobs by 32 % but cut energy by 26 %, similar to DVFS at 70 % frequency.
- Early-stage capping thus offers a gentle trade-off, making it attractive when slight slowdowns are acceptable.

Trade-Off Analysis

Both strategies confirm a **trade-off between runtime and energy**:
- Clock frequencies scaling yields proportional energy savings for predictable runtime increases.
- Power capping allows incremental power and energy reductions with minimal early slowdowns, then deeper cuts at greater caps.
- Choosing between them depends on whether maximum performance is vital (favor clock frequencies scaling) or flexible governors and finer control are desired (favor power capping).

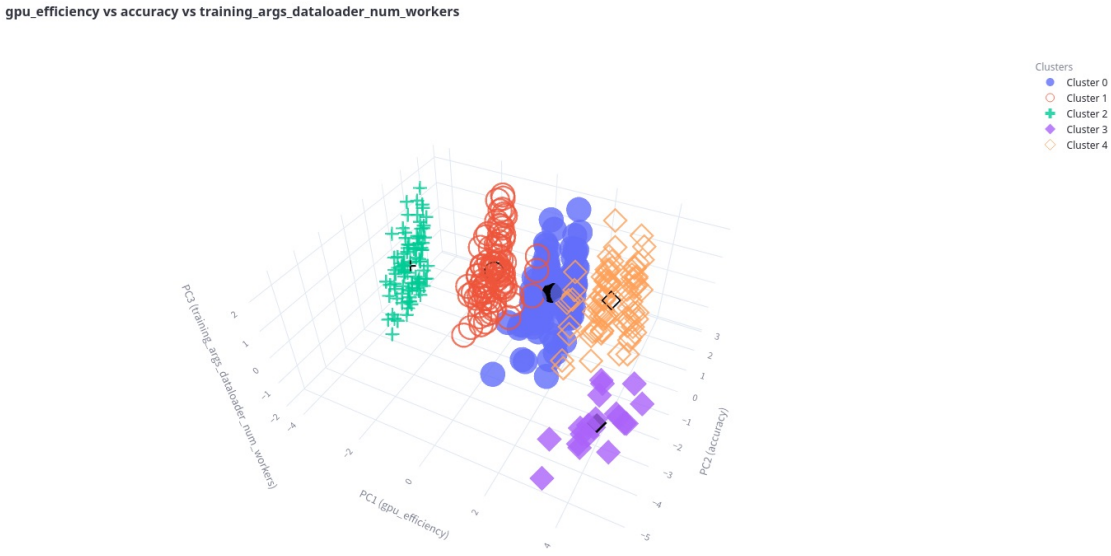
The results demonstrate that while runtime minimization remains a strong lever for energy reduction, hardware-level controls provide complementary knobs. Integrating clock frequencies scaling and power-capping into job schedulers can enable adaptive, energy-delay optimal execution.

Clusters Distribution and Relations Between Runtime, Energy Consumption and Accuracy

In this study, we investigate the relationship between energy consumption, model accuracy, and runtime, with the goal of identifying the most influential parameters. To begin, we perform clustering on the full feature vector, encompassing both the input parameters and the outcome metrics. We present here the data collected from the **Cali3**.

Parameters with Limited Impact on Execution Behavior

To assess the influence of different parameters, we applied parameter-result clustering and visualized the outcomes using PCA (Principal Component Analysis). This analysis revealed that certain parameters have minimal effect on the resulting energy and performance profiles, as illustrated in the following plot.



Cluster	Eval loss	Accuracy	Runtime	Batch size	Gradient accumulation steps	fp16	Dataloader workers	TPU cores	Energy (Wh)	CPU efficiency	GPU efficiency	Global efficiency
0	0.17	0.94	1,164.91	32	6.47	0.49	6.46	250.43	436.50	0.56	0.91	0.82
1	0.17	0.94	1,280.67	17.10	5.92	0.43	4.62	264.83	454.60	0.56	0.85	0.78
2	0.17	0.94	1,523.03	8	6.32	0.51	6.02	258.75	515.60	0.57	0.81	0.75
3	0.19	0.94	1,90.04	64	16	0.50	6	256	413.90	0.56	0.93	0.83
4	0.17	0.94	1,110.04	64	3.75	0.49	6.03	252.68	422.70	0.56	0.93	0.84

We observe that within each cluster, the mean values of certain parameters have no significant variation (**fp16, dataload workers, TPU cores**), indicating that these parameters exert minimal influence on the results.

Based on this insight, we conduct a second clustering analysis excluding these less influential parameters. This refined approach produces better-separated clusters, thereby facilitating clearer visualization and interpretation of the factors that drive energy efficiency.

Fine analysis

Our clustering analysis on the full feature set—incorporating evaluation loss, accuracy, runtime, batch size, and gradient accumulation steps—reveals distinct training profiles. The following cases emerge from the clustering results:

- **Fast but Less Accurate Configuration**:
  Some configurations (e.g., Cluster 3) employ larger batch sizes (64) paired with high gradient accumulation steps (16). These setups yield very short runtimes (around 1,090 seconds) but tend to show a higher evaluation loss (0.1895) and lower accuracy (0.9358). This trade-off indicates that, while the training proceeds rapidly, the accuracy of the model is compromised.

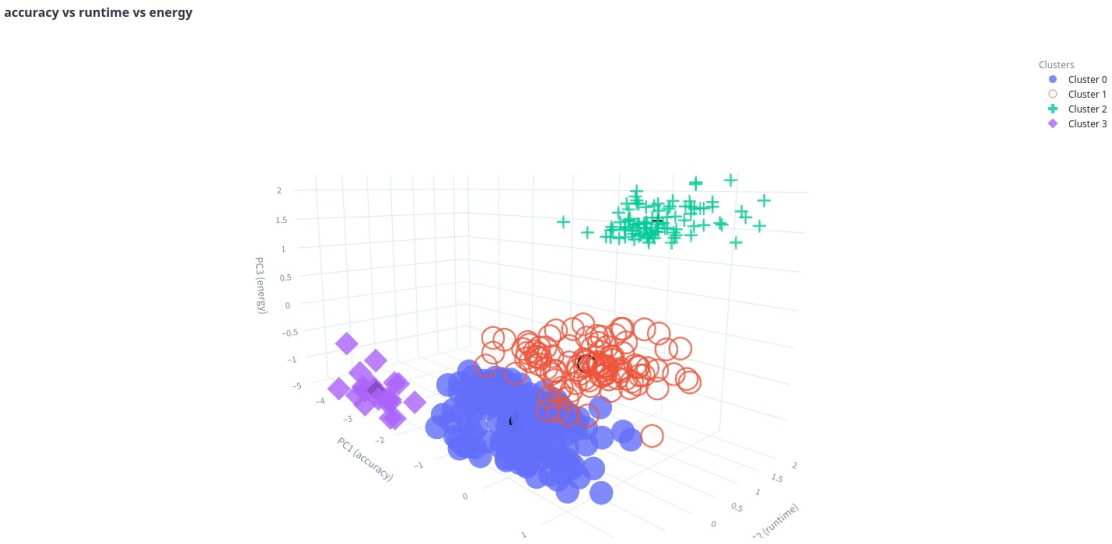
- **Looser Training with Smaller Batch Sizes**:
  Configurations with relatively small batch sizes (e.g., Cluster 1 with a batch size of 8) tend to use moderate gradient accumulation steps (around 6.32). Although these setups yield an extended runtime (approximately 1,523 seconds), the resulting evaluation loss is lower (0.1678) and accuracy is slightly higher (0.9436). This “looser” training regime appears to favor accuracy over speed.

- **Accurate Training with Balanced Settings**:
  A further balanced scenario is observed in configurations like Cluster 2, which employs an intermediate batch size (approximately 17) with moderate gradient accumulation (around 6). This setting achieves the best trade-off with the lowest evaluation loss (0.1662) and highest accuracy (0.9441) at a moderate runtime (around 1,281 seconds).

Regarding the gradient accumulation steps, our analysis indicates that while they play a role in influencing the

effective batch size and thus the update frequency, their impact is most evident when considered alongside the batch size parameter. In scenarios with high batch sizes, increases in gradient accumulation tend to further shorten runtime at the expense of accuracy. Conversely, moderate settings for both parameters tend to strike a favorable balance between convergence speed and model accuracy.

Below are the cluster distributions summarizing these findings:



Cluster	Eval loss	Accuracy	Runtime	Batch size	Gradient accumulation steps	fp16	Dataloader workers	TPU cores	Energy (Wh)	CPU efficiency	GPU efficiency	Global efficiency
0	0.17	0.94	1,165.66	32	6.41	0.49	6.39	253.25	436.60	0.56	0.91	0.82
1	0.17	0.94	1,523.03	8	6.32	0.51	6.02	258.75	515.60	0.57	0.81	0.75
2	0.17	0.94	1,281.20	16.93	5.98	0.42	4.67	261.95	454.60	0.56	0.85	0.78
3	0.19	0.94	1,90.04	64	16	0.50	6	256	413.90	0.56	0.93	0.83
4	0.17	0.94	1,110.04	64	3.75	0.49	6.03	252.68	422.70	0.56	0.93	0.84

These experimental results underscore the importance of tuning both batch size and gradient accumulation to achieve desired trade-offs between runtime speed and model accuracy. Faster training configurations tend to yield reduced accuracy, whereas more gradual training with smaller batch sizes leads to improved accuracy, albeit with increased runtime. This fine analysis reinforces the need for practitioners to carefully consider hyperparameter settings, particularly when aiming for energy-efficient and high-performance model training.

Discussion and Conclusion

This study introduces an energy measurement tool particularly adapted to the HPC-AI context, bringing a new dimension to training optimization by incorporating energy consumption alongside runtime and accuracy. Our results clearly demonstrate that training duration is a major contributor to total energy usage, as power is drawn continuously throughout the fine-tuning process. However, working solely on minimizing runtime is not sufficient. Optimizing training parameters based on the correlation between time and energy consumption is crucial.

One of the most influential parameters identified in our study is the **batch size**. For GPU hardware accelerators, this result was to be expected. Larger batch sizes tend to reduce training time due to increased per-step efficiency, thereby lowering total energy consumption. In contrast, smaller batch sizes extend runtime, resulting in higher cumulative energy usage, even though the power draw per step remains relatively stable. Our cluster analysis highlights that balancing batch size with gradient accumulation steps is critical for achieving optimal energy efficiency without sacrificing model accuracy. Notably, our experiments revealed that other parameters, such as numerical precision (fp16/fp32) and TPU cores, appeared to have little to no influence on energy consumption and runtime in our measurement context.

A key aspect of our work is **the quantification of the time-to-energy ratio, which provides a normalized metric for comparing different configurations**. This metric underlines that hardware efficiency alone is not the sole determinant of energy performance; training strategy and parameter tuning play an equally important role. In our experiments, a strong correlation between runtime and energy consumption was observed, confirming that focusing on time reduction is a viable approach to lowering energy expenditure. However, our findings stress that simply reducing runtime is not an end in itself, it must be balanced with achieving the desired accuracy and overall performance.

Although identifying configurations that reduce energy consumption by deliberately slowing down computation proved non-trivial, our results show that frequencies and power capping could offer a promising lever, particularly when aligned with the efficiency profile of a given workload. Fine-tuning power limits based on job characteristics may help lower energy use without proportionally increasing runtime. While this is not a broad avenue for optimization, it remains a niche path worth exploring in contexts where energy constraints are tight and workloads are well understood. We also intend to delve deeper into the software layers used. It is highly likely that opportunities for energy optimization exist within these layers.

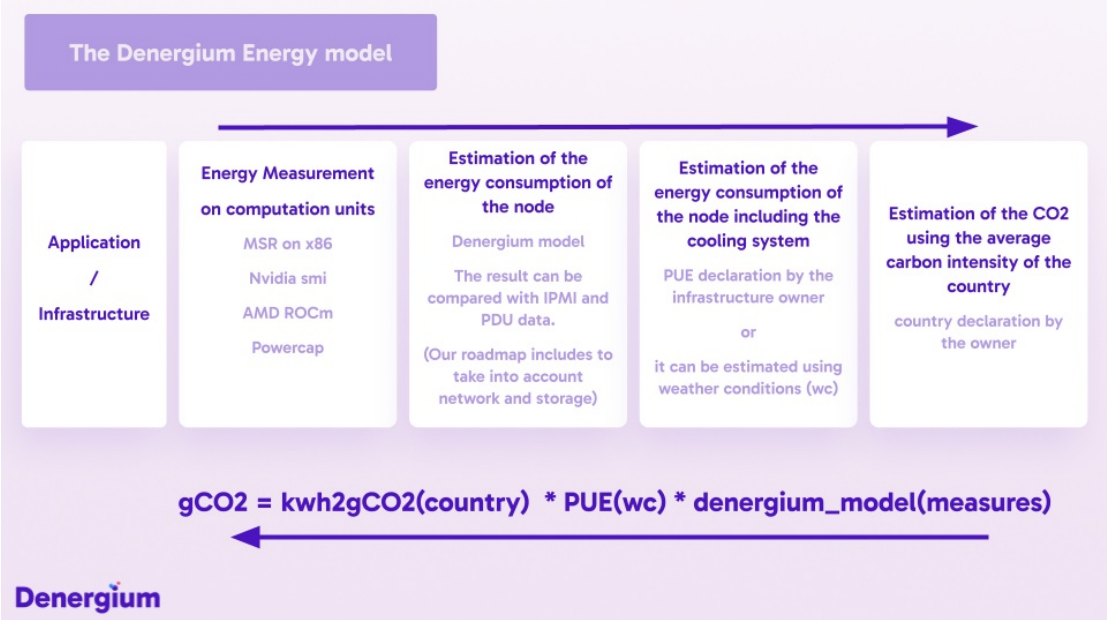
These insights lay the groundwork for our ongoing and future projects on optimization tools. By adopting a triadic constraint approach—considering accuracy, energy, and duration (which may also serve as a budget constraint), we aim to further refine the software-hardware matching process. Moving forward, we plan to extend our research to other application domains, such as numerical simulation, to broaden the impact of energy-aware optimization in diverse high-performance computing scenarios.

Tools like Hugging Face Accelerate or DeepSpeed offer efficient hyperparameter estimation and optimization, adjusting for hardware constraints like memory and computation. These tools simplify the tuning process, but they have limitations, such as relying on predefined search spaces and sometimes missing hardware-specific nuances. While they offer great value, achieving optimal performance still requires balancing automation with manual adjustments. One could imagine a utility that leverages these tools and adds an energy data component to help with hyperparameter tuning, viewed through the lens of a time-to-energy ratio.

While our experiments focused on fine-tuning a Transformer model, these insights should not extend, without caution, beyond AI into other deep learning and machine learning workloads, such as convolutional neural network training, reinforcement learning, or gradient-boosted decision trees, as well as traditional HPC tasks like large-scale numerical simulations, finite element analyses, and computational fluid dynamics. Each of these domains exhibits unique computation and memory patterns, varying model depths or solver algorithms, data dimensionality, and communication overhead, that shape their energy and performance profiles. Factors such as hardware choice (GPUs, TPUs, multi-core CPUs), scaling strategy (horizontal across nodes versus vertical within a node), and problem characteristics (mesh size, batch dimensions, or feature count) can all shift the optimal balance between runtime and power draw. Consequently, energy-aware tuning strategies must be adapted and validated for each workload family to ensure truly efficient, sustainable high-performance computing.

Throughout our study, energy consumption was measured at the CPU and GPU levels, offering reliable insights into consumption patterns as parameters varied. These measurements are sufficient for meaningful comparisons, but other infrastructure-level factors, like server type or cooling systems, can also influence the total energy footprint. For instance, the choice of datacenter itself can shift the energy balance.

To address this broader picture, Denergium has extended its energy model to include such infrastructural aspects.



Acknowledgments

This study was made possible by access to powerful computing resources. We gratefully acknowledge their invaluable support, which enabled us to conduct extensive experiments and in-depth analyses on state-of-the-art hardware platforms. Their commitment to advancing research and innovation in high-performance computing has been instrumental in deepening our understanding of energy efficiency and performance optimization.

Denergium is part of Meluxina’s Startup Program, which has provided access to advanced resources.

This work benefited from access to the computing resources of the “CALI 3” cluster. This cluster is operated and hosted by the University of Limoges. It is part of the HPC network in the Nouvelle-Aquitaine Region, financed by the State and the Region.

Experiments presented in this paper were carried out using the Grid’5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations.