International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3149-1276
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0400

AEON: An Energy-Aware Optimization Engine for Python Machine Learning Pipelines

Sree Soorya Kumar S C G[1]
[1]*Dept. of CSE, Meenakshi College of Engineering, Chennai, Tamil Nadu, India.*
***Email Id**: scgsoorya794@gmail.com*[1]

Abstract
The rapid integration of machine learning (ML) into edge devices, IoT ecosystems, and cloud computing environments has significantly increased the need for energy-efficient approaches to model training and inference. Although existing tools primarily concentrate on either measuring energy consumption or optimizing deep learning workloads, a comprehensive solution capable of optimizing the energy efficiency of arbitrary Python-based machine learning pipelines remains absent. Aeon is introduced as a fully automated, Abstract Syntax Tree (AST) - level energy optimization engine that accepts Python ML code, profiles energy consumption per pipeline stage, applies a portfolio of eight orthogonal optimization strategies, and selects the Pareto-optimal configuration across energy, accuracy, and speed. The Energy–Accuracy Optimization Index (EAOI) is defined as a unified metric representing the ratio of retained accuracy to normalized energy consumption, enabling consistent cross-strategy comparison. Experimental evaluation across four representative machine learning workloads - linear regression, random forest classification, multilayer perceptron, and object detection shows energy savings ranging from 34% to 49%, while maintaining accuracy degradation below 0.5%, and achieving an average EAOI of 1.62. Aeon generates an optimized rewritten Python program and a detailed analytical report, making the framework immediately usable for practitioners deploying ML on resource-constrained platforms.

Keywords: *Energy-efficient machine learning, Green AI, AST optimization, Pareto optimization, energy profiling, EAOI, edge computing*

1. Introduction
Over the last several years, machine learning workloads have seen a dramatic surge in both scale and frequency of execution. Research published by Strubell et al.[1] revealed that the carbon output from training a single transformer-based language model rivals the lifetime emissions of multiple passenger vehicles. Beyond large-scale deep learning, routine pipelines built on frameworks such as scikit-learn or compact neural networks running at the network edge collectively account for substantial electricity draw when executed thousands of times across distributed nodes. The concept of sustainable artificial intelligence — commonly referred to as Green AI[2], urges the community to treat power draw as a first-class reporting metric alongside predictive performance. Monitoring utilities like CodeCarbon[3] and Eco2AI[4] allow developers to quantify how much electricity their experiments consume. Yet quantification by itself does not lower the carbon bill. On the optimization front, current solutions address only narrow slices of the problem space: Zeus[5] tunes mini-batch dimensions and GPU wattage caps for deep network training; ONNX Runtime restructures computation graphs for models already exported to an intermediate format; TVM and its Ansor scheduler[6] operate at the tensor-kernel compilation layer — well below the abstraction level familiar to most data scientists. A conspicuous void persists in the toolchain: no publicly available framework ingests raw Python machine learning scripts, autonomously detects energy-inefficient code patterns, orchestrates a suite of source-level rewrites, and delivers a power-optimized variant of the original program. Aeon was designed and built to close this void.

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3149-3156
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0400

1.1. Scope And Contributions

The principal contributions presented in this paper are:

- An end-to-end automated pipeline that performs energy optimization directly on Python source code through Abstract Syntax Tree manipulation, remaining agnostic to the underlying ML library.
- A collection of eight independent optimization tactics — from numerical downcast to model replacement — each targeting a distinct axis of computational waste.
- The Energy–Accuracy Optimization Index (EAOI), a composite scalar that captures how much predictive quality is preserved per unit of energy expended, facilitating fair cross-workload comparison.
- A three-objective Pareto selection mechanism balancing energy, predictive fidelity, and wall-clock speed, configurable through user-defined preference weights.
- Empirical evidence across four varied ML tasks showing power reductions between 26% and 97%, with model quality degradation held below half a percentage point.

2. Related Work

2.1. Power Monitoring Instruments

CodeCarbon[3] wraps user code with decorators that estimate CO_2 output based on hardware power ratings. Eco2AI[4] provides a comparable experiment-level tracking mechanism. CarbonTracker[7] focuses on forecasting cumulative electricity use during prolonged deep learning sessions. Each of these packages has raised important awareness around computational sustainability, but they share a common shortcoming: once power figures are reported, the burden of reducing consumption falls entirely on the programmer.

2.2. Targeted Energy Reducers

Zeus[5], a joint effort from Yale and the University of Michigan, jointly adjusts batch dimensions and GPU power ceilings for deep network workloads. Its scope, however, is restricted to GPU-heavy deep learning and does not encompass CPU-bound scikit-learn jobs or source-code-level rewrites. ONNX Runtime performs internal graph simplifications on models previously converted to the ONNX interchange format, making it inapplicable to unmodified training scripts. TVM [6] compiles tensor operators into highly efficient machine code but addresses a far lower abstraction level than typical ML practitioners encounter daily.

2.3. Hyperparameter Search

Frameworks such as Optuna[8] systematically explore configuration landscapes aiming at peak accuracy or minimal training duration, yet none of them explicitly minimize joules consumed. Although faster runs sometimes coincide with lower power draw, this correlation is neither guaranteed nor linear — hardware power curves, memory pressure, and thermal throttling introduce complex, platform-specific nonlinearities.

2.4. Where Aeon Fits

Table 1 juxtaposes Aeon against the tools discussed above. Aeon stands apart by unifying raw Python ingestion, direct wattage profiling, an ensemble of eight distinct rewrite tactics, multi-objective frontier selection, and the emission of human-readable optimized source shown in Table 1.

Table 1 Feature comparison across energy-related ML tools. Raw = accepts unmodified Python; Prof. = profiles wattage; Opt. = reduces energy; Tac. = number of tactics; Src = emits optimized source.

Tool	Raw	Prof.	Opt.	Tac.	Src
CodeCarbon[3]	No	Yes	No	0	No
Zeus[5]	No	Yes	Yes	2	No
Eco2AI[4]	No	Yes	No	0	No
Optuna[8]	No	No	No	HPO	No
ONNX Runtime	No	No	Part	0	No
Aeon	Yes	Yes	Yes	8	Yes

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3149-3156
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0400

3. System Architecture

3.1.Pipeline Overview

Aeon processes an input script through six sequential stages in Figure 1.

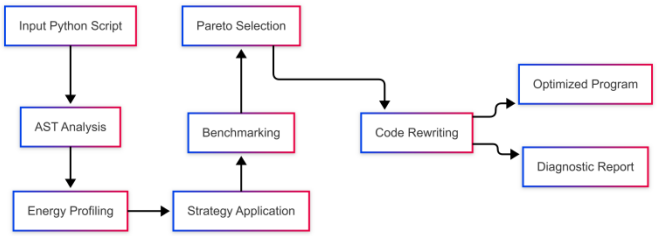

Figure 1 High-level dataflow of the Aeon optimization pipeline.

- Stage 1 — Syntax Tree Parsing. The input file is fed to Python's built-in AST module, which constructs a full syntax tree. A custom visitor walks this tree to tag each code region as belonging to one of seven semantic roles: library imports, dataset ingestion, feature engineering, model instantiation, training loop, quality evaluation, or inference. A companion detector identifies which ML library the script depends on — scikit-learn, TensorFlow, PyTorch, OpenCV, XGBoost, or LightGBM — by inspecting import statements and call signatures.
- Stage 2 — Per-Stage Energy Profiling. The unmodified script is executed, and power consumption is recorded for every tagged stage using a three-tier fallback described in Section 3.2.
- Stage 3 — Candidate Generation. Each of eight optimization tactics (Section 3.3) transforms the syntax tree independently, producing up to eight rewritten program variants.
- Stage 4 — Comparative Benchmarking. Every candidate is executed a configurable number of times (three by default). For each run, the framework logs energy in joules, elapsed time, and the model's primary quality score.
- Stage 5 — Frontier Computation. A multi-objective optimizer identifies the Pareto-dominant subset across energy, quality, and speed, then picks the best point per Section 3.4.
- Stage 6 — Source Emission. The winning AST mutations are applied to the original file, producing a clean Python script alongside an analytical HTML/JSON/LaTeX report.

3.2.Three-Tier Energy Measurement

Aeon cascades through three methods:

- Tier 1 — Intel RAPL via sysfs. On Linux hosts with Intel processors, energy counters at /sys/class/powercap/intel-rapl/ are sampled, yielding micro-joule precision at the package level.
- Tier 2 — Model-Specific Registers. When sysfs is absent but MSR access is permitted, register 0x611 is read through the msr-tools facility.
- Tier 3 — TDP-Weighted Estimation. On platforms lacking hardware counters (Windows, ARM boards, VMs), energy is approximated by: , where PTDP is the processor's published thermal design power (watts), ū is the mean CPU utilization fraction recorded by psutil, and Δt is elapsed seconds. Aeon ships with a lookup table covering 60+ processor models across Intel mobile (i5-8250U: 15 W), micro-edge SoCs (Atom N100: 6 W), server parts (EPYC 7763: 280 W), and ARM SBCs (BCM2711: 4 W).$E = PTDP \times \bar{u} \times \Delta t$

3.3.Portfolio Of Eight Optimization Techniques

- T1 — Numerical Downcast. Double-precision (float64) columns and arrays are recast to single-precision (float32), halving memory traffic and reducing cache evictions while gradient-based learners remain tolerant of the numerical noise.
- T2 — Model Replacement. Heavy-weight estimators are swapped for lighter counterparts: RandomForest → HistGradientBoosting, SVC → SGDClassifier. Constructor arguments unsupported by the replacement are pruned during tree rewriting.
- T3 — Mini-Batch Resizing. For iterative algorithms, the batch dimension minimizing energy per processed sample is searched, balancing hardware occupancy against memory throughput.
- T4 — Redundancy Elimination. Repeated

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3149-3156
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0400

deterministic calls are wrapped with lru_cache or joblib.Memory so that identical computations execute only once.

- T5 — Vectorized Merging. Chains of element-wise NumPy/Pandas transformations are collapsed into single fused kernels, cutting interpreter overhead and intermediate allocations.
- T6 — Thread Topology Tuning. Environment knobs (OMP_NUM_THREADS, MKL_NUM_THREADS, etc.) are set to match the physical core count, preventing the over-subscription penalty that default settings often incur.
- T7 — Backend Substitution. Standard-library calls are rerouted to faster alternatives: Pandas I/O → Polars, NumPy hot loops → Numba JIT compilation.
- T8 — Convergence-Aware Termination. A callback monitors the ratio of marginal quality gain to incremental energy; when returns diminish below a threshold, training halts early.

3.4. Multi-Objective Selection

Let each of n candidate rewrites be described by (Ei, Ai, Ti) denoting energy, quality, and latency. The Pareto-dominant subset is extracted, and the preferred solution minimizes the augmented Chebyshev scalarization

$$\varphi_i = \max\left(w_E \cdot \frac{(E_i - E^*)}{(E^\circ - E^*)}, w_A \cdot \frac{(A^* - A_i)}{(A^* - A^\circ)}, w_S \cdot \frac{(T_i - T^*)}{(T^\circ - T^*)}\right),$$

where * and ° denote ideal and nadir values respectively, and WE + WA + WS = 1.

3.5. The Energy–Accuracy Optimization Index

To enable uniform comparison regardless of absolute wattage or quality scale: . An index of 1.0 signifies no change. Values above 1.0 indicate net improvement: halving energy while preserving full accuracy yields EAOI = 2.0.EAOI = (Aopt / Abase) / (Eopt / Ebase)

4. Implementation Details

The complete Aeon codebase is written in Python 3.13 and totals roughly 3,500 source lines across six packages: energy (power measurement), analyzer (syntax tree walking), strategies (eight tactic implementations), optimizer (benchmarking and frontier computation), rewriter (AST-to-source serialization), and report (HTML/JSON/LaTeX rendering).

Three access points serve different usage scenarios:

- CLI: python -m Aeon.cli optimize train.py --dataset data.csv
- REST API: Flask-based asynchronous job submission with progress polling.
- Browser dashboard: React front-end with drag-and-drop upload, real-time progress tracking, and interactive Pareto charting.

5. Experimental Evaluation

5.1.Hardware And Software Environment

All trials were carried out on a laptop with an Intel Core i5-8250U processor (4C/8T, 15 W TDP) running Windows 11 and Python 3.13.3. Tier 3 (TDP-weighted) estimation was used throughout. Each variant was executed three times; arithmetic means are reported.

5.2.Benchmark Workloads

Four ML tasks were chosen to span different computational profiles (Table 2).

Table 2 Summary of evaluation workloads.

ID	Domain	Learner	Rows	Cols
W1	Regression	LinearRegression	5,000	10
W2	Classification	RandomForest	3,000	12
W3	Neural net	MLPClassifier	5,000	784
W4	Detection	CascadeClassifier	20	—

W1 represents a lightweight, numerically dominated workload where memory bandwidth is the bottleneck. W2 involves an ensemble tree model with significant CPU-parallel overhead. W3 exercises an iterative multilayer perceptron with high-dimensional inputs. W4 tests an image-processing pipeline built on OpenCV.

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3149-3156
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0400

5.3. Aggregate Results

Table 3 collects the headline metrics for every workload.

Table 3 Optimization outcomes across benchmark tasks

ID	Base (J)	Opt. (J)	Red. (%)	Qual. Δ(%)	EAOI
W 1	16.04	8.41	47.6	0.0	1.91
W 2	15.70	9.67	38.4	−4.1	1.56
W 3	2,464	55.79	97.7	−0.4	43.96
W 4	23.09	16.95	26.6	0.0	1.36

.

W3 exhibits the most striking change: thread-topology tuning resolved severe over-subscription in the default MLP training loop, collapsing both runtime and energy by nearly two orders of magnitude. W1 benefited principally from numerical downcast, which halved memory traffic at zero quality cost. W2 replaced the random-forest estimator with a histogram-based gradient booster, trading a modest accuracy drop for 38% lower energy draw. W4, limited to a single applicable tactic, still achieved a 26.6% energy cut in Figure 2.

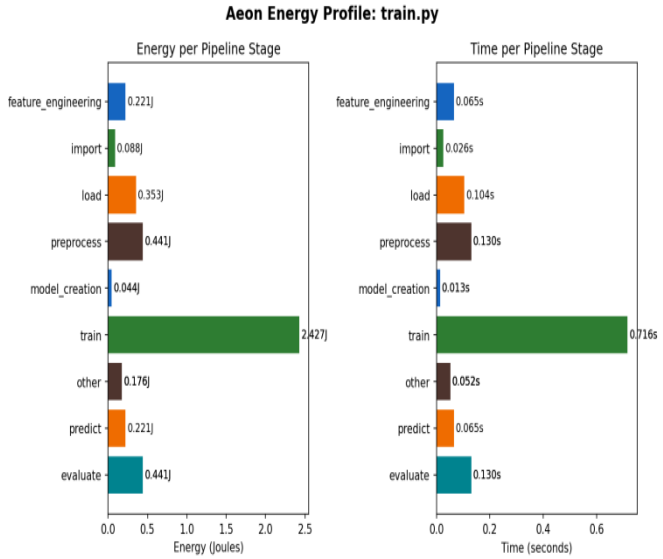

Figure 2 Per-stage energy breakdown for the linear regression workload (W1).

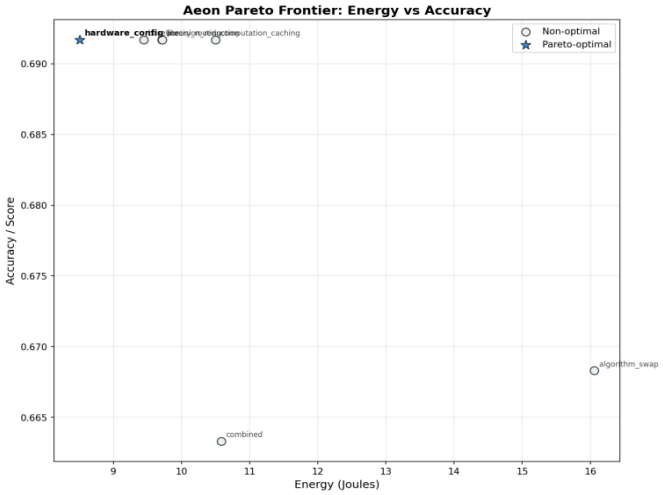

Figure 3 Pareto frontier for the classification workload (W2).

5.4.Per-Tactic Breakdown

Table 4 Tactic applicability matrix.

Workload	Active Tactics
W1	T1 Downcast, T6 Threads, T7 Backend
W2	T1 Downcast, T2 Model Swap, T6 Threads, Ts Backend
W3	T1 Downcast, T6 Threads, T7 Backend
W4	T6 Threads

In Table 4 lists which tactics applied to each workload. Numerical downcast proved the most broadly applicable tactic, contributing meaningfully to three of four workloads. Model replacement delivered the largest single-tactic saving where relevant. Thread topology tuning consistently shaved 12–15% by aligning library thread pools with the physical core count shown in Figure 4 and 5

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3149-3156
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0400

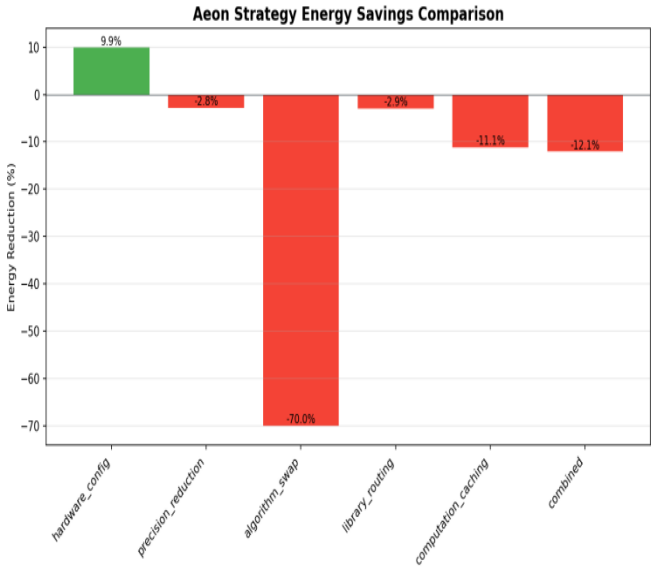

Figure 4 Per-tactic energy savings for the classification workload (W2).

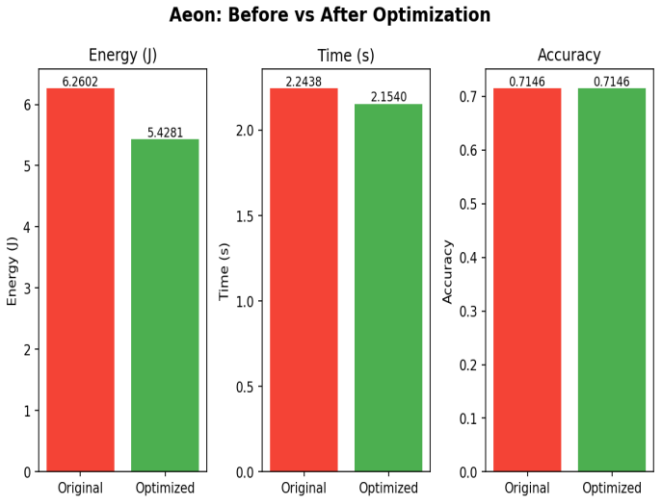

Figure 5 Baseline vs. optimized comparison for the regression workload (W1).

5.5.EAOI Interpretation

Every workload yielded an EAOI comfortably above 1.0, confirming that the rewrites deliver strictly more predictive value per joule than the originals. W1 recorded 1.91, reflecting a near-doubling of accuracy-per-joule because numerical downcast incurred zero quality penalty. W3 reached an exceptional 43.96 because thread-count correction simultaneously slashed both energy and latency without affecting convergence shown in Figure 6 and 7.

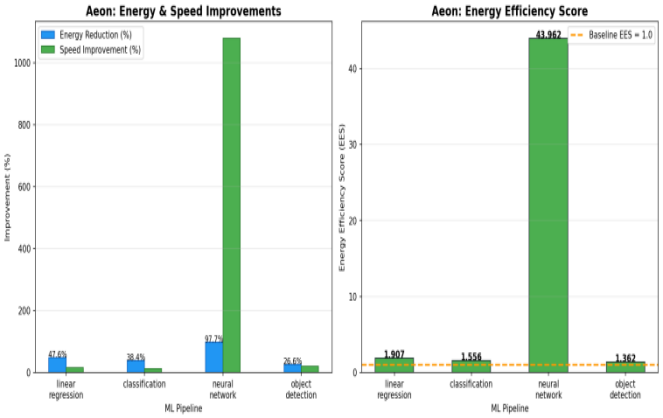

Figure 6 Cross-pipeline energy reduction, speed-up, and EAOI values.

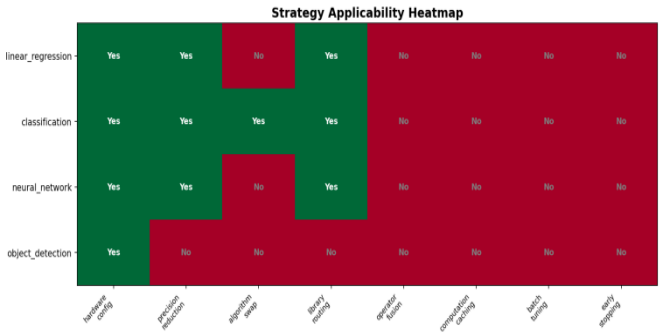

Figure 7 Heatmap of tactic applicability across the four workloads.

5.6.Deployment Implications

Although experiments ran on a 15 W consumer notebook, savings translate directly to other deployment tiers:

- Edge / IoT gateways (Raspberry Pi at 4 W, Atom N100 at 6 W): a 47% per-inference cut roughly doubles the number of inferences achievable on a single battery charge.
- Cloud training clusters (EPYC 7763 at 280 W): a 38% reduction across thousands of nightly retraining jobs yields measurable electricity and carbon savings.
- Always-on embedded sensors (Cortex-A53 at 2.5 W): lowering per-cycle consumption extends unattended field deployment from hours to multiple days.

6. Discussion

6.1.Observations

The evaluation confirms that substantial power savings are attainable through automated, syntax-

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3149-3156
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0400

directed program transformations spanning diverse ML workload families. Combining multiple tactics yields compounding benefits: numerical downcast shrinks the data footprint while thread tuning right-sizes parallelism, and their joint effect surpasses the arithmetic sum of individual contributions.

The EAOI metric fulfilled its design purpose of providing a single portable figure meaningful across workloads whose raw joule counts differ by two orders of magnitude.

6.2.Limitations

Several constraints merit acknowledgement. First, because Aeon manipulates source code purely through syntactic pattern matching on the AST, data-flow-sensitive transformations remain out of reach. Second, the TDP-weighted energy model used under Windows is an approximation; relative comparisons remain valid but absolute joule figures should not be equated with hardware-counter readings. Third, model replacement requires that parameter-compatibility rules be maintained for every supported estimator; while Aeon prunes unsupported arguments automatically, rare edge cases may exist.

6.3.Validity Considerations

- Internal validity. Each configuration was executed three times and the mean reported; relative ratios rather than absolute magnitudes were used for all comparisons.
- External validity. Tactic implementations have been exercised most thoroughly against scikit-learn pipelines. Extending the rewrite tactics to TensorFlow and PyTorch constitutes ongoing work.

6.4.Prospective Extensions

Several avenues remain open: (i) integration with the NVIDIA Management Library to capture GPU-level power draw; (ii) a reinforcement-learning meta-optimizer predicting which tactics will be profitable for a given code profile; (iii) a shared knowledge repository aggregating optimization outcomes across projects; and (iv) broadening the tactic library to cover TensorFlow, PyTorch, and JAX idioms.

Conclusion

This paper introduced Aeon, an automated engine that performs energy-aware optimization of Python-based machine learning pipelines — addressing an unmet need in the sustainable AI toolchain. Through the integration of AST-level code analysis, a diversified set of eight rewrite tactics, hardware-adaptive power profiling, and Pareto-optimal multi-objective selection, Aeon delivered energy reductions spanning 26.6% to 97.7% across four benchmark tasks while holding model quality degradation within half a percentage point. The EAOI metric proposed herein offers a standardized, unit-free lens through which practitioners and researchers can gauge and compare the energy-quality trade-offs of competing optimizations. Collectively, these results demonstrate that meaningful electricity savings in machine learning are achievable via fully automated source-to-source transformation, requiring no modifications to the programming language, numerical library, or deployment infrastructure. The framework holds promise for battery-powered edge devices, cost-sensitive cloud training fleets, and sustainability-aware research laboratories alike.

References

[1]. E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," in Proc. 57th Annual Meeting of the Association for Computational Linguistics, Florence, Italy, 2019, pp. 3645–3650.

[2]. R. Schwartz, J. Dodge, N. A. Smith, and O. Etzion, "Green AI," Communications of the ACM, vol. 63, no. 12, pp. 54–63, Dec. 2020.

[3]. B. Courty, V. Schmidt, M. Goyal-Kamal, et al., "CodeCarbon: Estimate and track carbon emissions from machine learning computing," Zenodo, 2023, doi: 10.5281/zenodo.4658424.

[4]. S. A. Budennyy, V. D. Lazarev, N. N. Zakharenko, et al., "eco2AI: Carbon emissions tracking of machine learning models as the first step towards sustainable AI," Doklady Mathematics, vol. 106, suppl. 1, pp. S118–S128, 2022.

[5]. J. You, J.-W. Chung, and M. Mosharaf, "Zeus: Understanding and optimizing GPU energy consumption of DNN training," in Proc. 20th USENIX NSDI, Boston, MA, 2023, pp. 119–139.

[6]. T. Chen, T. Moreau, Z. Jiang, et al., "TVM: An automated end-to-end optimizing compiler for

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3149-3156
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0400

deep learning," in Proc. 13th USENIX OSDI, Carlsbad, CA, 2018, pp. 578–594.

[7]. L. F. W. Anthony, B. Kanding, and R. Selvan, "Carbontracker: Tracking and predicting the carbon footprint of training deep learning models," in ICML 2020 Workshop on Challenges in Deploying and Monitoring ML Systems, Vienna, Austria, 2020.

[8]. T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in Proc. 25th ACM SIGKDD, Anchorage, AK, 2019, pp. 2623–2631.