Tu(r)ning AI Green: Exploring Energy Efficiency Cascading with Orthogonal Optimizations

Saurabhsingh Rajput, Mootez Saad, Tushar Sharma
Dalhousie University
Halifax, Canada
{saurabh,mootez,tushar}@dal.ca

***Abstract*—AI's exponential growth intensifies computational demands and energy challenges. While practitioners employ various optimization techniques, that we refer as "knobs" in this paper, to tune model efficiency, these are typically afterthoughts and applied in isolation without understanding their combined effects on energy efficiency. The goal of this exploratory empirical study is to emphasize on treating energy efficiency as the first-class design consideration by demonstrating how strategic knob selection across five AI pipeline stages (data, model, training, system, inference) creates cascading efficiency gains. We evaluate 30 experimental variants on ModernBERT, an encoder-only architecture, examining individual techniques and their orthogonal combinations. Results shows that model pruning provides the highest single-knob energy savings (up to 84.6%), while orthogonal combinations reduce energy consumption by up to 94.6% while preserving 95.95% of baseline F1 score. This work provides actionable frameworks for informed green AI that balance efficiency, performance, and environmental responsibility in AI systems.**

***Index Terms*—Green AI, Software Engineering, Sustainable Software Development**

I. Introduction

The exponential growth of generative Artificial Intelligence (AI) has fundamentally transformed modern technology landscapes, with its market size growing at an extraordinary rate of 34.90% annually [1]. Software Engineering (SE) stands at the forefront of this AI revolution, integrating AI into critical tasks from code completion to generation, testing, and deployment. While this advancement drives innovation, it comes with a hidden cost. Modern AI models consume enormous computational resources, resulting in significant energy usage and carbon emissions. In response, **Green AI** initiatives advocate making energy and compute efficiency core evaluation criteria for AI research, alongside traditional metrics such as accuracy. These initiatives focus on developing energy-efficient algorithms, optimizing model architectures, and reducing computational requirements.

Practitioners now employ numerous optimization techniques such as distillation, pruning, and quantization to enhance efficiency. However, these approaches are often applied piecemeal, optimizing one stage of the AI development pipeline while neglecting others. As Cruz *et al.* [2] highlight, this isolated optimization can be counterproductive; for example, reducing training energy at the expense of higher inference costs.

This challenge is compounded by Jevon's paradox [3]: continued increase in aggregated energy consumption despite efficiency gains due to increased adoption. While recent models achieve better performance-per-parameter—DeepSeek-Coder-Instruct-6.7B matched CodeLlama-Instruct-34B's performance [4]—their collective energy footprint paradoxically expands due to widespread deployment. This underscores the need for an integrated, software-engineering-driven agenda. Our work answers this call, aligning with Cruz *et al.*'s roadmap [2] to guide the development of truly sustainable AI systems.

In this work, we demonstrate how strategic usage of Green AI optimization techniques across five development stages—data, model architecture, training, system design, and inference—creates cascading efficiency gains. Through rigorous experimentation, we show that orthogonal combinations (targeting distinct resource constraints without mutual interference) of optimization techniques yield compounded benefits, reducing energy consumption between 4.6% and 94.6% while maintaining model performance (-4.05 to $+0.12\%$ F1 score). The study emphasizes transforming energy efficiency from an afterthought to a core design consideration, empowering practitioners to achieve sustainable AI without compromising the capability of their AI pipeline.

II. The Optimization-Efficiency Conundrum

The computational intensity of state-of-the-art AI models continues to escalate with training requirements doubling approximately every 5.3 months—outpacing hardware efficiency gains [5]. This acceleration is evident in model scaling from GPT-3's 175 billion parameters to Deepseek-R1's 671 billion and beyond [5], with each leap increasing energy demands.

A. Where Current Practice Falls Short

Despite the availability of techniques for AI models computational efficiency, energy consumption remains high. This stems from a *compute-centric* optimization mindset: practitioners prioritize reducing hardware utilization (*e.g.,* GPU size) and maintaining accuracy to run large-scale models, treating energy savings as a possible byproduct rather than a primary design goal. As shown in Figure 1, numerous optimization techniques are proposed. However, developers consider them as an afterthought, not proactively. This highlights two critical aspects. First, energy consumption constitutes a fundamental

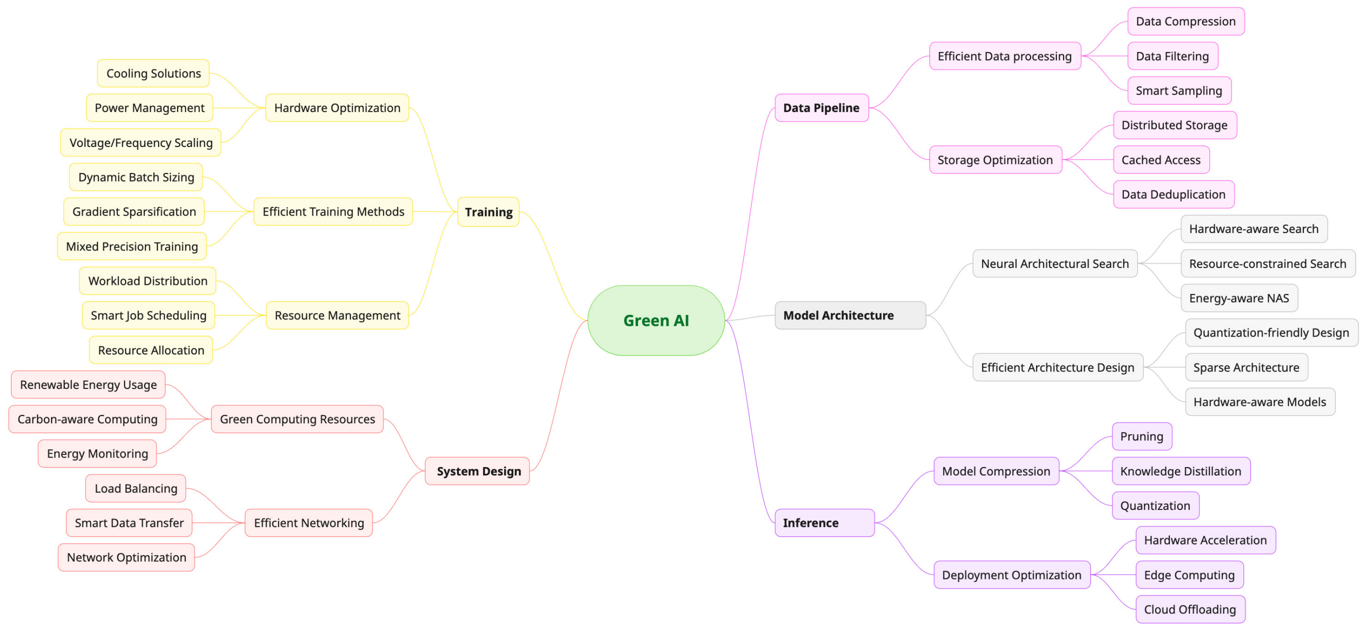

Fig. 1. Green AI techniques: An overview of green AI techniques across different development stages. Our experiments (Table II) evaluate techniques from each stage: **Data Pipeline** (efficient data processing, storage optimization), **Model Architecture** (sparse architectures via pruning, efficient architecture design), **Training** (mixed precision training, resource management, hardware optimization), **System Design** (efficient networking and computing resources), and **Inference** (model compression, deployment optimization). This enables assessment of individual technique impacts and cross-stage orthogonal combinations.

optimization dimension that needs to be treated as a first-class citizen similar to accuracy and other performance metrics. Second, though these techniques may individually lead to some energy efficiency, their careful combinations may exhibit orthogonal effects that may lead to significant efficiency gain.

B. Green Practices as First-class Citizens

Most optimization techniques deployed today originate from *compute-centric* experiments where energy efficiency is not considered as an objective. **Our vision emphasizes energy consumption as a fundamental design constraint across all development stages, enabling practitioners to *tune their AI green* through strategic combinations of orthogonal optimization techniques/knobs.** We propose transforming sustainability from an afterthought to a core design principle by:

1) embedding energy awareness during *design, development, and execution* of AI pipelines,
2) combining knobs *orthogonally* for amplified efficiency, and
3) explicitly balancing accuracy-energy tradeoffs.

Table I detail how implementing this proposal across five critical development stages (shown in Figure 1) creates cumulative energy savings while maintaining performance.

Takeaways. AI development prioritizes accuracy over energy while applying scattered compute optimizations. Embedding energy as first-class objective from the outset propagates efficiency throughout the pipeline.

III. Research Approach

Research Method:: We conduct an empirical experimental study to operationalize the vision from Section II. Our investigation addresses two research questions, scoped for an encoder-only architecture:

- **RQ1:** What are the individual energy and performance impacts of optimization techniques applied in isolation across the five pipeline stages?
- **RQ2:** How do strategic combinations of orthogonal optimizations affect overall system efficiency?

Section II identified critical shortcomings: practitioners apply techniques as afterthoughts in isolation, neglecting energy efficiency and combinatorial effects. Figure 1 illustrates techniques across five-stages, yet their systematic application remains underexplored.

We evaluate each optimization *knob* across two dimensions: (1) *knob isolation*—single-knob variants establishing baseline characteristics, and (2) *knob combination*—multi-knob bundles integrating orthogonal optimizations (*e.g.,* pruning with FP16 and compilation) targeting distinct constraints. This validates our hypothesis: strategically curated optimizations deliver cascading benefits, while poorly curated combinations yield diminished effects. These experiments test whether opportunities identified in Figure 1 deliver measurable improvements when systematically explored.

Experimental Setup: All experiments fine-tune ModernBERT-base, a modern transformer underlying contemporary AI systems, on BigVul—a vulnerability-classification dataset (217, 000 samples, 348 projects) providing sufficient scale to observe energy-patterns while

TABLE I
CURRENT PRACTICES IN AI MODEL LIFECYCLES AND OUR PROPOSED GREEN PRACTICES FOR EACH PHASE.

Stage	Current Practices	Green Practice
Data Pipeline	Current data practices emphasize quantity over quality, with companies amassing enormous datasets by scraping the Internet. Such web-scale data collection often leads to redundant storage and repeated processing. Modern LLMs are trained on trillions of tokens; for instance, LLAMA-3 was trained on approximately 15 trillion tokens. Despite diminishing returns, the industry trend continues to favor larger datasets [6].	Effective data management practices must shift to a more resource-aware approach. Instruction-tuning LLMs on curated, high-quality prompts can yield comparable or superior performance [7]. Techniques such as Smart-sampling, model input trimming, efficient tokenization, and strategic-filtering can reduce computational loads. Energy-conscious data pipelines, smart sampling, deduplication, and locality-aware storage mitigate unnecessary data transfers and processing.
Model Architecture	Current architectural trends favor larger models with billions of parameters, following the *bigger is better* scaling law [8]. The intent is to prioritize performance gains through increased model capacity and computation.	Sparse architectures (*e.g.,* pruned models), reducing parameter counts, using activation functions such as ReLU, adopting hardware-aware designs such as flash attention, quantization-friendly architectures, and energy-aware Neural Architecture Search (NAS).
Training	Contemporary training practices focus on achieving state-of-the-art performance through computationally-intensive approaches. Models are typically trained on massive GPU clusters, with repeated training runs. For example, GPT-4's training is estimated to consume over 2×10^{10} petaFLOPS [5]. This stage emphasizes brute-force approaches.	Mixed precision training [9], dynamic batch sizing [10], gradient sparsification [11], adaptive resource allocation, renewable-energy-aware distributed training, smart batch sizing, and energy-aware hyperparameter optimization.
System Design	Current system architectures for AI-enabled applications prioritize performance and compute scaling, treating energy efficiency as an afterthought. Load balancing ignores energy consumption, and autoscalers lack power monitoring or carbon-aware scheduling [12]. The absence of standardized energy-monitoring creates significant barriers.	Carbon-aware scheduling, comprehensive monitoring tools for real-time resource tracking, and leveraging predictive analytics for resource management (*e.g.,* power capping [10], DVFS, etc).
Inference	Current inference deployments predominantly prioritize low latency and high throughput, often neglecting energy efficiency by replicating model instances globally. This is pronounced with large models like GPT-4. Since inference is approximately 80% of total AI computational demands [13], post-training optimization is crucial.	Dynamic pruning, quantization, knowledge distillation, KV caching context-sensitive model selection (*e.g.,* mixtures-of-experts), intelligent pruning, shared inference infrastructure, and real-time energy monitoring.

remaining computationally feasible for 30 variants. The 22 single-knob variants were identified primarily from HuggingFace [14], ensuring findings apply to widely adopted practitioner techniques. Each variant ran thrice on a clean system, with no extraneous processes. Experiments are executed on a NVIDIA-RTX-5000-GPU (32-GB) with an AMD-EPYC-9554P CPU. CodeCarbon[1] monitors energy at one-second intervals. Hyperparameters remain fixed; only optimization configurations vary. The replication package is available [15].

IV. CASCADING EFFECTS ACROSS THE PIPELINE

Table II presents comprehensive results for all 30 experimental variants across both families, revealing how individual techniques and their strategic combinations impact energy consumption, model performance, and execution time.

A. RQ1: Knob Isolation Effects

To understand the isolated impact of each optimization *knob*, we activated them individually on the ModernBERT baseline (V_0, 0.512 kWh, 0.994 F1), resulting in 22 distinct variants. Table II summarizes these results, highlighting several critical insights listed below.

- **Training dominates energy (90-92%), making it the highest-priority target.** Aggressive pruning (V_{21}–V_{22}) primarily reduces training energy. Data Pipeline optimizations (V_8) create cascading benefits downstream. Inference represents 8-10% experimentally but dominates production workloads. System Design choices (V_{11}) affect execution across stages.
- **Structural sparsity yields significant efficiency gains.** Layer pruning stands out among performance-preserving optimization techniques, providing substantial energy savings with minimal impact on model performance. Variants V_{21} (20-layer top pruning) and V_{22} (20-layer bottom pruning) achieve energy reductions of 84.6% and 84.5%, respectively, with negligible accuracy trade-offs (Δ F1: -0.08% and $+0.09\%$). The energy savings from the pruning scale nearly linearly with the number of layers removed, reaching an optimal trade-off at 16-layer pruning (V_{19}, V_{20}: energy reduction of 68.4%, Δ F1 $<$ 0.11%).
- **Compiler and precision optimizations provide**

[1]https://github.com/mlco2/codecarbon

TABLE II

Comprehensive Analysis of Optimization Variants. Stage indicators: 🟪 Inference, 🟥 Data, 🟧 System, ⬛ Model, 🟨 Train. Energy columns measure: Data (data loading), Token (tokenization), Load (model loading), Train (training), Save (checkpoint saving), Infer (inference/evaluation), Total (aggregate consumption in kWh). Performance (Perf.) shows F1 score changes. Time columns show execution time in seconds. Green highlighting marks best-performing (lowest energy, fastest time) values per column; orange marks worst-performing values. V30 (Optimal) combines techniques from green-highlighted variants across columns to demonstrate maximum cascading potential.

Variant	Description	Stage(s)	**Energy** (kWh)								**Perf.**	**Time** (s)		
			Data ($\times10^{-5}$)	Token ($\times10^{-6}$)	Load ($\times10^{-5}$)	Train ($\times10^{-1}$)	Save ($\times10^{-5}$)	Infer ($\times10^{-2}$)	Total ($\times10^{-1}$)	ΔTotal (%)	ΔF1 (%)	Total ($\times10^{3}$)	Infer ($\times10^{2}$)	ΔInfer (%)
V_0	Baseline (ModernBERT)	⬜⬜⬜⬜⬜	2.79	4.91	2.47	4.68	3.84	4.44	5.12	0.0	+0.00	7.92	5.17	+0.0
V_1	Gradient Checkpointing	⬜⬜🟨⬜⬜	2.68	4.51	2.55	6.25	4.03	4.45	6.70	+30.7	−0.03	10.4	5.18	+0.1
V_2	LoRA PEFT	⬜⬛🟨⬜🟪	3.29	3.76	3.52	0.59	1.21	1.62	0.80	−84.3	−4.00	1.7	2.35	−54.5
V_3	Quantization	⬜⬛🟨🟧🟪	4.75	542	5.22	4.41	0.76	4.38	4.85	−5.3	−0.06	11.6	6.43	+24.4
V_4	Tokenizer Optimization	🟥⬜⬜🟧🟪	3.19	4.79	2.40	4.68	3.68	4.45	5.13	+0.1	+0.11	7.93	5.19	+0.3
V_5	Power-Limit (100 W)	⬜⬜⬜🟧⬜	7.86	5.01	2.42	5.92	3.47	5.56	6.48	+26.5	−0.05	20.5	13.8	+167.6
V_6	Optimizer Tuning	⬜⬜🟨⬜⬜	4.88	4.57	2.40	4.44	3.72	4.45	4.89	−4.6	+0.003	7.63	5.18	+0.3
V_7	FP16 Training	⬜⬛🟨🟧⬜	2.88	4.49	2.36	3.73	3.87	3.46	4.07	−20.5	−0.014	6.61	4.21	−18.5
V_8	Sequence-Length Trim	🟥⬜🟨🟧⬜	8.61	3, 530	2.95	2.48	4.01	2.16	2.73	−46.6	−0.38	4.44	2.55	−50.6
V_9	Inference Engine	⬜⬜⬜🟧🟪	3.31	5.5	2.92	4.66	112.9	0.39	4.71	−8.11	−0.18	7.44	0.55	−89.2
V_{10}	Data-Loader Pin-Mem	🟥⬜🟨⬜⬜	9.61	4.93	2.43	3.73	3.60	3.49	4.08	−20.4	−0.02	6.64	4.26	−17.6
V_{11}	Torch Compile	⬜⬜🟨🟧🟪	5.58	4.43	2.40	2.77	4.10	3.81	3.15	−38.4	−0.01	4.76	4.45	−13.9
V_{12}	Attention Optimization	⬜⬛🟨⬜🟪	5.92	4.91	2.38	3.46	4.03	2.87	3.75	−26.8	+0.04	5.8	3.39	−34.5
V_{13}	Layer Pruning (4 Top)	⬜⬛🟨🟧🟪	2.91	5.31	2.57	3.98	3.30	3.70	4.35	−15.0	+0.10	6.61	4.30	−16.7
V_{14}	Layer Pruning (4 Bottom)	⬜⬛🟨🟧🟪	8.55	4.72	2.64	3.98	3.57	3.68	4.35	−15.2	−0.10	6.60	4.27	−17.3
V_{15}	Layer Pruning (8 Top)	⬜⬛🟨🟧🟪	7.79	4.80	2.48	3.10	2.70	2.94	3.39	−33.8	+0.072	5.3	3.41	−34.1
V_{16}	Layer Pruning (8 Bottom)	⬜⬛🟨🟧🟪	5.37	5.23	2.50	3.10	2.71	2.92	3.39	−33.8	−0.11	5.3	3.37	−34.8
V_{17}	Layer Pruning (12 Top)	⬜⬛🟨🟧🟪	2.98	4.82	2.55	2.40	2.74	2.18	2.62	−48.9	+0.12	3.94	2.51	−51.4
V_{18}	Layer Pruning (12 Bottom)	⬜⬛🟨🟧🟪	11.1	4.89	2.56	2.40	2.67	2.19	2.62	−48.8	−0.04	3.95	2.52	−51.2
V_{19}	Layer Pruning (16 Top)	⬜⬛🟨🟧🟪	11.4	4.44	2.59	1.48	2.43	1.43	1.62	−68.3	+0.11	2.55	1.63	−68.5
V_{20}	Layer Pruning (16 Bottom)	⬜⬛🟨🟧🟪	3.21	5.14	2.51	1.47	2.38	1.42	1.62	−68.4	+0.05	2.54	1.61	−68.9
V_{21}	Layer Pruning (20 Top)	⬜⬛🟨🟧🟪	3.11	4.98	2.50	0.73	1.84	0.65	0.79	−84.6	−0.08	1.14	0.74	−85.6
V_{22}	Layer Pruning (20 Bottom)	⬜⬛🟨🟧🟪	3.65	4.90	2.51	0.73	1.60	0.65	0.80	−84.5	+0.09	1.14	0.747	−85.5
V_{23}	Attn+Pin-Mem+Opt8+GradAcc	🟥⬛🟨🟧🟪	3.39	4.54	2.37	4.09	4.21	3.72	4.46	−12.9	+0.11	6.5	4.08	−21.0
V_{24}	Inf+GradCkpt+LoRA+FP16	⬜⬛🟨🟧🟪	6.24	4.53	3.19	0.64	107.0	0.37	0.69	−86.6	−4.05	1.69	0.54	−89.5
V_{25}	GradAcc+FP16+Checkpoint	⬜⬛🟨🟧⬜	4.89	4.38	2.36	6.44	3.97	4.50	6.89	+34.4	+0.12	10.3	4.96	−4.0
V_{26}	Prune12B+Seq-Len+Compile	🟥⬛🟨🟧🟪	5.75	551	2.67	0.89	2.59	0.92	0.99	−80.8	−0.26	1.45	1.21	−76.6
V_{27}	Torch Compile + FP16	⬜⬛🟨🟧🟪	3.31	4.49	2.44	2.41	4.28	3.56	2.77	−46.0	+0.03	4.1	4.18	−19.1
V_{28}	Prune12B + Compile + FP16	⬜⬛🟨🟧🟪	5.02	4.59	2.56	1.17	2.67	1.74	1.35	−73.7	−0.07	2.1	2.10	−59.3
V_{29}	Attn+Pin-Mem+Opt8	🟥⬛🟨🟧🟪	2.73	6.20	2.70	3.26	3.64	2.90	3.56	−30.52	−3.29	5.53	3.42	−33.72
V_{30}	Optimal	🟥⬛🟨🟧🟪	9.3	4.3	372.1	0.20	247.8	0.25	0.27	**-94.6**	−4.0	**0.54**	0.37	**-92.69**

performance-preserving efficiency wins. Techniques such as `torch.compile` (V_{11}) significantly reduces energy by 38.4% and wall-time by approximately 40%, with only a negligible accuracy impact (Δ F1: −0.01%). Similarly, FP16 training (V_7) delivers a 20.5% energy reduction while shortening runtime 16.6%, further emphasizing the practical benefits of these low-overhead methods. Sequence-length trimming (V_8) halves padding overhead, trimming energy 46.6% and time 43.9% at a moderate –0.38% F1 cost. LoRA (V_2) is frugal with (–85.3% energy, –78.6% time) but sacrifices 4 % F1, making it suitable only when accuracy budgets allow.

- **Memory-focused optimizations can be counterproductive.** Not all memory-centric optimizations improve overall energy efficiency. Gradient checkpointing (V_1) incurs an energy penalty of 30.7%, primarily due to the computational overhead introduced by recomputation. Similarly, static power limiting (V_5), although intuitively beneficial, results in a 26% energy increase coupled with a substantial latency rise of 168%. Quantization (V_3) yields only modest energy savings (5%) but significantly extends runtime (46.9%), suggesting applicability around resource-constrained devices.

Figure 2 provides practitioners with an actionable roadmap for optimization decisions by mapping the trade-off between inference time and energy consumption relative to the baseline configuration. This visualization serves three critical functions for production deployment:

1) **Prioritization guidance**: Variants in the bottom-left quadrant (green) simultaneously reduce both energy and latency, representing accuracy-neutral first choices for immediate implementation (*e.g.,* layer pruning in V_{17}–V_{22}).
2) **Tradeoff identification**: Upper-left quadrant variants reduce energy but increase runtime (*e.g.,* quantization in V_3), suitable for batch processing or edge deployments where latency is less critical than energy conservation.
3) **Risk mitigation**: Right-half variants indicate counter-productive optimizations that degrade key metrics (*e.g.,* static power limiting in V_1, V_4, V_5), serving as warnings to avoid these configurations.

Practitioners can leverage this visualization by first implementing bottom-left quadrant optimizations as foundational efficiency improvements, then progressively stacking additional knobs along favorable trajectories to amplify gains. When evaluating new configurations, teams should benchmark against V_{30}'s empirical lower bound (marked by the yellow star) to assess optimization potential. Finally, optimization efforts should conclude once application-specific accuracy or latency budgets are reached, ensuring resource-efficient deployment without unnecessary tuning.

Takeaways: Structural pruning provides the highest single-knob energy savings. Memory-saving techniques introducing additional computation often backfire, increasing energy consumption. Instead of power-limiting, use workload-aware methods or precision optimization to avoid increased latency and energy.

B. RQ2: Knob Combination Dynamics

Next, we evaluate eight multi-knob bundles (V_{23}–V_{29}) along with an optimal *oracle* combination (V_{30}) to explore potential synergistic and antagonistic effects from combined optimizations. The results, that we summarize below, support our hypothesis that strategically curated optimizations can lead to significant cascading efficiency gains.

- **Synergistic combinations amplify efficiency.** Combinations of orthogonal knobs—those targeting distinct resource constraints such as computational intensity, arithmetic precision, and compiler optimization—consistently yield energy savings beyond the sum of their isolated effects. Specifically,
  - V_{26} compounds savings, cutting energy by 80.8% with a minor (ΔF1: -0.26%) accuracy cost, exceeding individual savings.
  - V_{28} realizes a significant 73.7% energy saving while maintaining model accuracy with minimal degradation (ΔF1: -0.07%).
  - V_{27} delivers a notable 46.0% energy reduction and slightly improves accuracy (ΔF1: +0.03%).
- **Antagonistic combinations reduce or reverse gains.** Conversely, combining optimizations that target similar resource bottlenecks often results in diminished or even negative efficiency outcomes. Specifically,
  - V_{25}) increases energy consumption by 34.4%, due to excessive recomputation overhead.
  - V_{29} yields a poor trade-off: 30.5% energy savings for a major ΔF1 loss (-3.29%).
- **Stage sequencing amplifies cascading efficiencies.** Early-stage optimizations such as V_8 significantly reduce the downstream computational load, magnifying the overall efficiency gains in later stages. For example, combining sequence-length trimming with pruning and compilation (V_{26}) redistributes energy consumption across pipeline stages, resulting in reduction in total system energy.
  The *oracle* V_{30}, combining optimal strategies, achieved maximal savings (ΔEnergy: -94.6%, ΔTime: -93.2%). This proves cross-stage optimizations can yield near-total savings and preserve core functionality, despite a 4% F1 trade-off.
- **Optimization Strategies Along the Efficiency Frontier** The energy-time tradeoff visualization in Figure 2 reveals three pathways for practitioners to navigate efficiency optimizations:
  - *Accuracy-sensitive pathway:* Layer pruning (16–20 layers) combined with attention optimization delivers substantial energy savings (68%–85%) with negligible accuracy loss (ΔF1 $< 0.11\%$). This approach occupies the desirable bottom-left quadrant in Figure 2, simultaneously reducing both energy and latency.
  - *Maximum-efficiency pathway:* Combining aggressive pruning, FP16 precision, and compiler optimizations achieves maximum energy savings (73%–94%) with modest to moderate accuracy trade-offs (ΔF1: -0.07% to -4.00%).
  - *A Middle Ground:* Inference engine optimization (V9) complements both these pathway well, delivering dramatic time reductions (89.2%) with minimal accuracy impact (0.18% F1 reduction), making the combination ideal for latency-sensitive applications where performance preservation is critical. It give's modest energy gains (-8.11%); however, if combined with variants from other stages as demonstrated in V_{24} and V_{30} it achieves up to 92.69% time

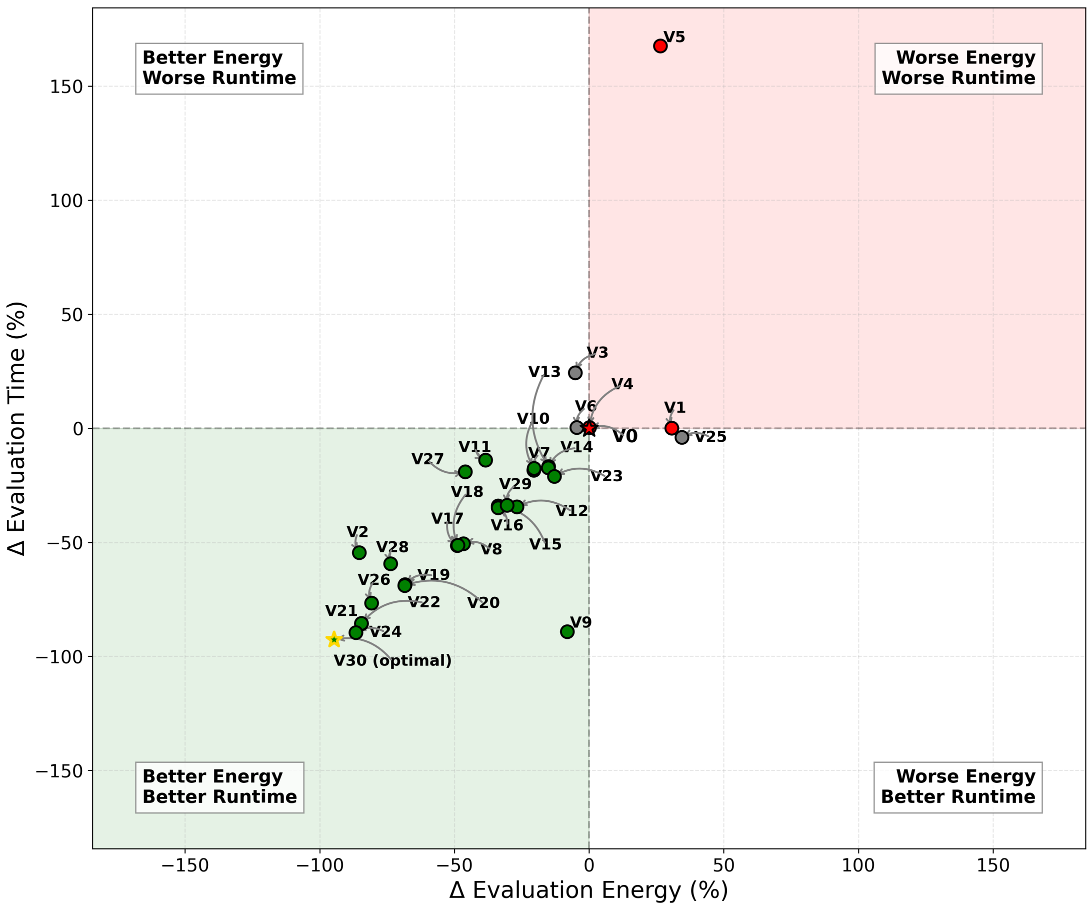

Fig. 2. Inference Energy vs. Inference Time Trade-off. The plot shows the percentage change in evaluation energy consumption (Δ Energy) versus the percentage change in inference time (Δ Time) relative to the baseline variant (V_0). Points in the bottom-left quadrant represent improvements in both energy efficiency and runtime, while points in the top-right quadrant indicate worse performance in both metrics. The baseline variant (V_0) is marked with a red star at the origin. Points are color-coded based on their performance relative to the baseline: green for improvements in both metrics, red for worse performance in both metrics and gray for mixed results.

reduction, along with upto 94.6% energy savings. This pathway pushes toward the empirical lower bound (yellow star in Figure 2), making it suitable for large-scale deployments where aggregate resource savings outweigh marginal performance differences.

C. Practitioner Guidelines

Table II's stage indicators map variants to Figure 1, enabling practitioners to identify optimization opportunities. Our replication package [15] enables adaptation to specific models and tasks through `config` file modifications.

Stage prioritization: Target Training first (90-92% of energy). Data Pipeline optimizations create cascading downstream benefits. Inference becomes critical in production deployments.

Quick wins: Prioritize bottom-left quadrant variants (Figure 2) for immediate gains. Avoid counterproductive techniques like gradient-checkpointing (V1: +31% energy) and static power-limiting (V5: +27% energy, +168% latency).

Deployment strategies: Accuracy-sensitive applications should use V17-V22 ($< 0.11\%$ F1 impact). Latency-critical systems benefit from inference engine optimization (V9) with cross-stage combinations. Edge deployments may accept quantization's (V3) runtime cost (47% increase) for reduced memory-footprint. Batch processing prioritizes dual energy-time reductions like V8 or V21-V22.

Takeaways: Combining orthogonal optimization knobs significantly amplifies energy efficiency. Avoid combining similar resource-targeted knobs, as it often negates potential gains. Sequence early-stage optimizations strategically to create cascading efficiency improvements downstream.

V. Threats to Validity

Internal Validity: Each variant ran three times on isolated system with no background processes. CodeCarbon monitored energy per second with consistent hardware. All techniques used standard APIs (PyTorch, HuggingFace) with verified correctness.

External Validity: Findings are specific to ModernBERT (encoder-only architecture), BigVul (classification task), and our hardware. Decoder-only models employ different mechanisms (causal attention, auto-regressive generation, KV caching) that may respond differently to optimizations. While the orthogonality principle—techniques targeting independent resource constraints create cascading benefits—should transfer broadly, specific percentages are architecture-dependent. Generative tasks may exhibit different patterns. Cross-architecture validation with decoder-only models, vision transformers, and diverse tasks remains essential future work.

Construct Validity: CodeCarbon measures operational energy excluding embodied manufacturing carbon. One-second sampling captures patterns in multi-hour runs. Measurements cover GPU/CPU energy per Green AI practice.

Conclusion Validity: Three repetitions establish consistency, with large effect sizes (up to 94.6% reduction) indicating genuine patterns. The cascading principle is demonstrated across multiple orthogonal combinations spanning different stages. We tested representative combinations from hundreds of possibilities; practitioners should evaluate specific configurations for their contexts. Figure 1 presents Green AI techniques organized by stage and orthogonality—not exhaustive; comprehensive enumeration requires systematic mapping beyond this work's scope.

VI. Final Remarks

Our experiments on encoder-only architectures demonstrate that strategically combining orthogonal optimizations achieves 94.6% energy reduction while preserving 95.95% F1 score through cascading efficiency gains. Realizing this potential requires coordinated action: industry adopting multi-knob optimization, researchers developing energy-aware frameworks, and policymakers establishing supportive regulations. Through collaboration, we can align AI advancement with environmental sustainability, treating sustainability as a catalyst to harness AI's potential while minimizing its footprint.

References

[1] World Bank, "Generative artificial intelligence." http://dx.doi.org/10.1596/39959, 2023. [Accessed 14-01-2025].

[2] L. Cruz, J. P. Fernandes, M. H. Kirkeby, S. Martínez-Fernández, J. Sallou, H. Anwar, E. B. Roque, J. Bogner, J. Castaño, F. Castor, A. Chasmawala, S. Cunha, D. Feitosa, A. González, A. Jedlitschka, P. Lago, H. Muccini, A. Oprescu, P. Rani, J. Saraiva, F. Sarro, R. Selvan, K. Vaidhyanathan, R. Verdecchia, and I. P. Yamshchikov, "Greening ai-enabled systems with software engineering: A research agenda for environmentally sustainable ai practices," 2025.

[3] B. Alcott, "Jevons' paradox," *Ecological economics*, vol. 54, no. 1, pp. 9–21, 2005.

[4] D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. K. Li, F. Luo, Y. Xiong, and W. Liang, "Deepseek-coder: When the large language model meets programming – the rise of code intelligence," *CoRR*, vol. abs/2401.14196, 2024.

[5] J. Sevilla and E. Roldán, "Training compute of frontier ai models grows by 4-5x per year." https://epoch.ai/blog/training-compute-of-frontier-ai-models-grows-by-4-5x-per-year, 2024. Accessed: 2025-10-23.

[6] J. Cheng, M. Marone, O. Weller, D. Lawrie, D. Khashabi, and B. V. Durme, "Dated data: Tracing knowledge cutoffs in large language models," in *First Conference on Language Modeling*, 2024.

[7] C. Zhou, P. Liu, P. Xu, S. Iyer, J. Sun, Y. Mao, X. Ma, A. Efrat, P. Yu, L. Yu, *et al.*, "Lima: Less is more for alignment," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[8] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, *et al.*, "Training compute-optimal large language models," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, pp. 30016–30030, 2022.

[9] S. Narang, G. Diamos, E. Elsen, P. Micikevicius, J. Alben, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, *et al.*, "Mixed precision training," in *Int. Conf. on Learning Representation*, 2017.

[10] J. You, J.-W. Chung, and M. Chowdhury, "Zeus: Understanding and optimizing gpu energy consumption of dnn training," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pp. 119–139, 2023.

[11] X. Lin, Y. Liu, F. Chen, X. Ge, and Y. Huang, "Joint gradient sparsification and device scheduling for federated learning," *IEEE Transactions on Green Communications and Networking*, vol. 7, no. 3, pp. 1407–1419, 2023.

[12] A. Radovanović, R. Koningstein, I. Schneider, B. Chen, A. Duarte, B. Roy, D. Xiao, M. Haridasan, P. Hung, N. Care, *et al.*, "Carbon-aware computing for datacenters," *IEEE Transactions on Power Systems*, vol. 38, no. 2, pp. 1270–1280, 2022.

[13] V. Avelar, P. Donovan, P. Lin, W. Torell, and M. A. T. Arango, "The AI disruption: Challenges and guidance for data center design," *Artificial Intelligence in Medicine*, vol. 138, 2023.

[14] D. Suhoi, "Efficient Deep Learning: a comprehensive overview of optimization techniques." https://huggingface.co/blog/Isayoften/optimization-rush, 2024. Accessed: 15-01-2025.

[15] S. Rajput, M. Saad, and T. Sharma, "How to Tune Your AI Green?." https://github.com/SMART-Dal/tuning-green-ai-pipelines, June 2025.