

DISSECTING TRANSFORMERS: A ‘CLEAR’ PERSPECTIVE TOWARDS GREEN AI

000
001
002
003
004
005 **Anonymous authors**
006 Paper under double-blind review
007
008
009
010
011
012
013
014
015
016
017
018
019
020
021
022
023
024
025
026
027
028
029
030
031
032
033
034
035
036
037
038
039
040
041
042
043
044
045
046
047
048
049
050
051
052
053

ABSTRACT

The rapid adoption of Large Language Models (LLMs) has raised significant environmental concerns. Unlike the one-time cost of training, LLM inference occurs continuously at a global scale and now dominates the AI energy footprint. Yet, most sustainability studies report only coarse, model-level metrics due to the lack of fine-grained measurement methods, treating energy efficiency more as an afterthought than as a primary objective. We present the first fine-grained empirical analysis of inference energy across core components of transformer architecture. We propose a novel methodology, **Component-Level Energy Assessment via Repeated sampling (CLEAR)**¹, to overcome temporal mismatch between microsecond(μ s) scale component execution and monitoring of millisecond(ms) scale energy sensors. Using CLEAR, we evaluate 15 models spanning four distinct architecture types and consistently keep component-wise energy variance below **9.5%** while capturing more than **90%** of the model’s total energy as individual components. Our empirical analysis reveals that Attention blocks consume significantly more energy per floating-point operation (FLOP), indicating that energy consumption is not proportionally aligned with FLOP counts. This shows that FLOPs alone fail to capture the true energy cost at a component level. Our findings establish detailed component-level energy baselines and provide insight as an initial step to build energy-efficient transformer models through component-level optimizations.

1 INTRODUCTION

Large language models (LLMs) such as GPT-4, Gemini, and Claude have transformed natural language processing, but their environmental costs are increasingly concerning. Using ChatGPT alone has been estimated to produce 8.4 tons of CO₂ annually, over twice the average individual’s yearly footprint (Earth.org, 2023), equivalent to driving a gasoline car for more than 20,000 miles (U.S. Environmental Protection Agency, 2023) or the carbon sequestration of 300 mature trees (EcoTree, 2023). Beyond emissions, inference consumes significant energy and water, Google reports that a single Gemini query uses up to nine seconds of television viewing, with an associated reliance on water-intensive cooling that strains local supplies (News, 2025; Google, 2025). While training and finetuning of LLMs is a computationally heavy and energy intensive process, it occurs infrequently. Inference, by contrast, happens continuously at a massive scale, with GPT models serving hundreds of millions of queries daily (TechTarget, 2025; AceCloud, 2024). This makes reduction of per inference energy consumption a primary objective for model’s performance optimization.

Current research focuses on model-level energy consumption, enabling high-level comparisons across systems (Alizadeh and Castor, 2024; Sánchez-Mompó et al., 2025a). However, such aggregate reporting obscures which architectural components e.g., Attention, or MLP blocks, drive energy usage. Fine-grained energy measurement is essential for identifying energy-intensive components, enabling targeted optimizations, and informing algorithm–hardware co-design for sustainable deployment.

¹Code: <https://anonymous.4open.science/r/CLEAR-D487>

In this paper, we introduce CLEAR, a methodology for fine-grained energy measurement of individual component blocks of the Transformer architecture during inference. Unlike prior work limited to model-level reporting, our approach decomposes transformers into constituent components, such as embeddings, layer Normalization, Attention, and feed-forward MLP and measures the energy consumed by each component. This allows us to establish detailed energy baselines and to compare consumption patterns across model architectures, sizes, input lengths, and floating point precisions. A central challenge arises from the temporal granularity mismatch between component execution and energy sensor monitoring as the transformer sub-operations complete within microseconds, whereas energy sensor provides power updates at tens of milliseconds. To address this, CLEAR employs an amplification strategy that stabilizes energy measurements. This design circumvents limitations of current monitoring infrastructure and achieves component-level isolation at a granularity previously inaccessible. Our contributions can be summarized as follows

1) We propose a novel methodology, CLEAR (Component-Level Energy Assessment via Repeated sampling), to overcome temporal mismatch between execution of microsecond (μs) scale components and milisecond (ms) scale energy sensors. CLEAR is deviced to measure energy consumption of fine-grained components of Transformer architecture.

2) Using CLEAR, we measure component-level energy consumption for 15 models across different model architectures with high consistency and completeness and perform an empirical analysis across different input token lengths and floating point precisions.

3) Through our empirical analysis, we observe that energy consumed per FLOP of computation varies significantly across components, with Attention Block exhibiting the highest energy consumed per FLOP. Empirically, we observe that energy consumption of components can be decomposed into fixed overheads and FLOP-dependent costs enabling future research to estimate

2 RELATED WORK

The environmental footprint of large language models (LLMs) has become a central research concern, with many studies quantifying the carbon emissions of training and inference. LLMCarbon Faiz et al. (2024) provides one of the most comprehensive analyses to date, modeling the end-to-end footprint of both dense and mixture-of-experts architectures across training, inference, experimentation, and storage. Vidur Özcan et al. (2025) complements this with GPU-based simulations, showing how batch size, sequence length, and parallelism affect inference efficiency. Fernandez et al. (2025) extend this to real workloads, distinguishing prefill and autoregressive decoding, and demonstrate that optimizations such as speculative decoding, kernel compilation, and continuous batching can cut energy use by up to 73%. Broader benchmarks, like *How Hungry is AI?* Jegham et al. (2025), evaluate energy, water, and CO₂ footprints across hardware platforms, while the BLOOM case study Luccioni et al. (2023) was among the first to track emissions during the training and inference of a 176B parameter model. Collectively, these efforts have established robust methodologies for characterizing the environmental impact of LLMs, but they report energy only at the aggregate model level due to the absence of methods for fine-grained measurement, leaving open the question of how energy is distributed across internal components

To improve accessibility, lightweight tools have emerged. CodeCarbon estimates emissions from GPU/CPU usage and regional carbon intensity, while Carbontracker Anthony et al. (2020) adds real-time monitoring and early prediction of training costs. These tools improve transparency but remain oriented toward system-level aggregation. Recent fine-grained approaches like Rajput et al. (2024) introduced FECOM, profiling TensorFlow APIs via static instrumentation to build one of the first framework-level energy datasets. Pinnock et al. (2025) proposed EDGEPROFILER, an analytical framework for lightweight LLMs on devices such as Raspberry Pi and Jetson Nano, showing that quantization can reduce memory use by 70% and energy by 50%, but that I/O bottlenecks often dominate latency. Hugging Face’s AI Energy Score Luccioni and collaborators (2025) benchmarks over 160 models on 10 tasks, reporting GPU energy use across preprocessing, prefill, and decode stages.

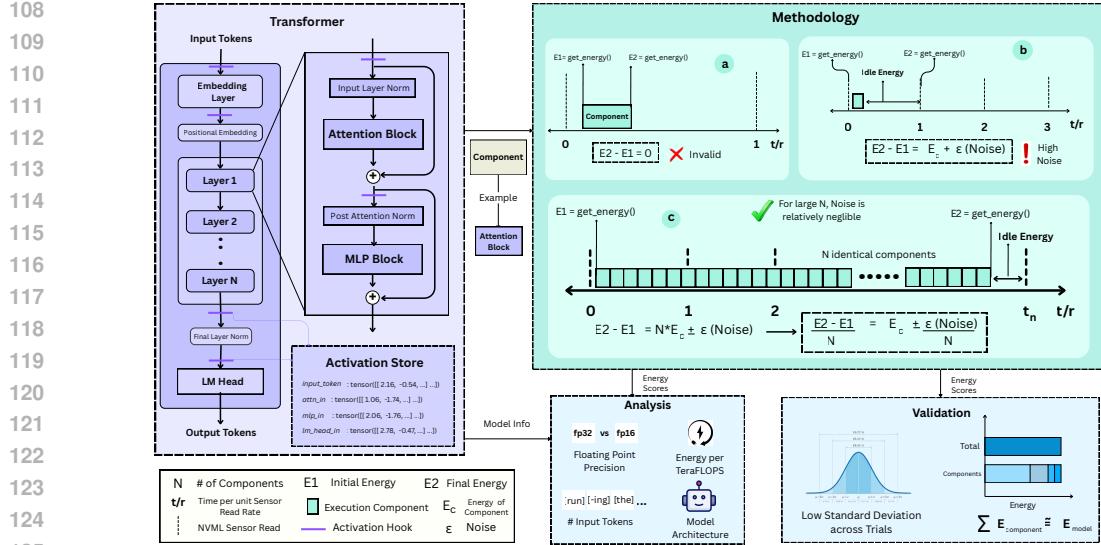


Figure 1: CLEAR pipeline: (1) Store forward-pass activations in Activation Store, (2) Measure per-component energy consumption by isolating each component (e.g., Attention Block) and replaying stored activations, (3) Validate for consistency and completeness, and (4) perform Empirical energy analysis across models.

Overall, existing studies neither provide a detailed empirical analysis of fine-grained, component-level energy consumption nor offer a reliable way to measure energy at such a small scale. Moreover, prior work often relies on FLOPs as a proxy for energy use, despite its limitations. This paper aims to close these gaps by introducing a method to accurately measure and analyze energy consumption at the component level.

3 METHODOLOGY

3.1 ACTIVATION STORE

Our methodology, CLEAR, targets component-level energy measurement by focusing on key computational primitives common to most transformer-based models as illustrated in Figure 1. These include an **Attention** block that captures token-level dependencies, feed-forward **MLP** blocks for dense nonlinear transformations, normalization blocks (Norm.), the **Embedding Layer** which maps discrete tokens into continuous vector spaces and the final **Language Modeling Head** (LM Head) that project hidden representations back to the vocabulary space for output generation. To enable component-wise energy profiling, we insert forward hooks at key points of the computation graph,

$$\mathcal{A} = \{\text{attn_in}, \text{mlp_in}, \text{lm_head_in}, \text{layer_norm_in}\dots\},$$

and capture the input activations at each hook. During a single forward pass, the hooks record for every component $c \in \mathcal{A}$ the corresponding activation tensor a_c for all tokens,

$$\mathbf{A} = \{a_c \mid c \in \mathcal{A}\}, \quad a_c \in \mathbb{R}^d.$$

The Activation Store \mathbf{A} (shown in Figure 1) serves as a cache of activations that allows isolated re-execution of individual components under identical input statistics, enabling fine-grained measurement of energy consumption. Refer to Appendix A for details about Transformer architecture.

3.2 AMPLIFICATION STRATEGY

Accurately measuring the energy consumption of individual components in a transformer-based LLM is challenging because their execution time is typically much shorter than the

162 sampling period of GPU power sensors. For example, NVIDIA’s NVML has a sensor read
 163 rate of about 20 to 50 ms. As shown in Methodology of Figure 1, the temporal mismatch
 164 leads to two distinct sources of error:

165 In case (a), when the component completes execution within a few microseconds, entirely
 166 between two sensor samples, the monitoring sensor cannot update its reading in time due to
 167 which the observed energy is reported as zero:

$$E_2 - E_1 = 0, \quad (1)$$

168 even though the true component energy is non-zero. This results in a complete underestimation
 169 of the component’s energy usage.

170 In case (b), if we supposedly measure energy after every sensor reading to capture the
 171 component’s energy consumption, the result remains highly noisy. This is because the
 172 measurement inevitably includes a significant amount of idle energy drawn by CUDA,
 173 making it hard to separate the true component energy. Consequently, when the execution
 174 only partially overlaps with a sensor’s sampling window, the observed energy is recorded as
 175

$$E_2 - E_1 = E_c \pm \varepsilon, \quad (2)$$

176 where E_c is the component’s actual energy consumed and ε represents noise.

177 To address above challenges, we adopt an amplification strategy, illustrated in Methodology (c)
 178 of Figure 1. As individual transformer components often complete the execution within 10–
 179 100 μ s, their energy consumption remains highly noisy to NVML’s coarse sampling window.
 180 The goal is to minimize noise (ε) in the component energy measurement to obtain reliable
 181 readings. This noise may arise from the model’s idle energy consumption or from inherent
 182 errors in the sensor measurements. To achieve this, we repeatedly execute each component
 183 back-to-back on the cached activations, with no gap between runs. This scales the effective
 184 runtime so that the total energy of the repeated executions outweighs the idle background
 185 consumption, making the noise comparatively negligible.

186 Concretely, for each component c with cached input a_c , we measure the energy before and
 187 after N consecutive executions:

$$E_c^{\text{tot}} = \text{MeasureEnergy} \left(\sum_{i=1}^N c(a_c) \right). \quad (3)$$

188 The per-execution energy can then be obtained by averaging the total measured energy:

$$\hat{E}_c = \frac{E_{\text{end}} - E_{\text{start}}}{N} = \frac{E_c^{\text{tot}}}{N} \pm \frac{\varepsilon}{N}, \quad (4)$$

189 where ε denotes the measurement noise. By increasing N , the duration of the aggregated
 190 workload extends to hundreds of milliseconds which is long enough for NVML’s power
 191 sensor to capture it while the noise term ε/N diminishes proportionally, yielding significantly
 192 reliable per-component energy estimates. (Refer Algorithm 1)

193 We repeat the amplified measurement for T trials with a brief pause between runs to let
 194 the sensor reset, taking Average and Standard Deviation across trials to further smooth out
 195 sensor noise and make the per-component energy estimate more reliable.

204 **Algorithm 1** CLEAR - Component Level Energy Assessment via Repeated sampling

205 **Require:** Model \mathcal{M} , components \mathcal{C} , input x , iterations N , trials T

206 1: Register forward hooks on $\{c \in \mathcal{C}\}$ to capture activations

207 2: $\mathbf{A} \leftarrow \text{ForwardPassAndCache}(\mathcal{M}, x)$

208 3: **for** each $c \in \mathcal{C}$ **do**

209 4: **for** $t = 1$ to T **do**

210 5: $E_c^{\text{tot}, (t)} \leftarrow \text{MeasureEnergy} \left(\sum_{i=1}^N c(a_c) \right)$

211 6: $\hat{E}_c^{(t)} \leftarrow \frac{E_c^{\text{tot}, (t)}}{N} \pm \frac{\varepsilon}{N}$

212 7: **end for**

213 8: $\bar{E}_c \leftarrow \frac{1}{T} \sum_{t=1}^T \hat{E}_c^{(t)}$

214 9: **end for**

215 10: **return** $\{\bar{E}_c \mid c \in \mathcal{C}\}$

216 3.3 VALIDATION
217

218 Since no existing literature has yet provided the energy consumption of individual components
219 within Transformer architectures, the goal is to validate our methodology along two key
220 dimensions 1) Consistency across trials and 2) Completeness of captured energy

$$222 \text{StdDev}(\bar{E}_c) \rightarrow 0, \quad \bar{E}_{\text{model}} \approx \sum_{c \in \mathcal{C}} \bar{E}_c.$$

225 1) A standard deviation close to 0 indicates that repeated component-level energy measure-
226 ments remain consistent across trials demonstrating high precision in energy measurements
227 by CLEAR. 2) The near-equality between the total measured model energy and sum of
228 its per-component energies demonstrates that CLEAR is able to capture the component’s
229 energy usage in a comprehensively.

230
231 4 EXPERIMENTAL DETAILS
232233 4.1 HYPERPARAMETER & HARDWARE SPECIFICATIONS
234

235 As part of our experimental protocol, we evaluate two floating-point precisions, FP32 and
236 FP16, while varying the input sequence length across 8, 32, 64, 96, and 128 tokens to study
237 scaling effects. Each configuration is run for a fixed set of 20 trials ($T = 20$) to capture
238 variability and validate precision. Assume FP16 precision unless explicitly stated

239 We also investigate the effect of the repetition count N on reliability of the energy readings. As
240 seen in Fig. 2, increasing repetitions N yields more stable readings and reduces measurement
241 failures i.e. cases where the recorded energy for a trial is spuriously 0 mJ due to sensor
242 granularity. Based on this analysis, we set the repetition count $N = 10,000$ for measurements
243 of small components with execution time of order $100 \mu\text{s}$ and $N = 1,000$ for energy
244 measurements of the full model, balancing measurement accuracy with computational cost.

245 Experiments were conducted on NVIDIA Ada-Lovelace GPUs (RTX 5000 Ada, RTX 6000
246 Ada), which incorporate third-generation RT cores and fourth-generation Tensor Cores
247 supporting mixed-precision operations such as FP8 with sparsity NVIDIA (2023); PNY
248 (2023). The RTX 5000 Ada provides 12,800 CUDA cores, 32 GB ECC GDDR6 memory, and
249 a board power of 250 W, while the RTX 6000 Ada offers 18,176 CUDA cores, 48 GB ECC
250 GDDR6 memory, and 300 W board power. However, the NVML interface, which typically
251 updates power readouts only every 20–50 ms Anonymous (2023), introduced limitations in
252 resolving microsecond-scale component execution, thereby requiring the CLEAR methodology
253 for fine-grained energy attribution. Code is available here.

254
255 4.2 METRICS256 4.2.1 ENERGY CONSUMPTION
257

258 The energy consumed by each model component is measured in milliJoules (mJ), matching
259 the 0.8 mJ precision of the NVML sensor used. For validating our methodology, we define
260 two complementary metrics, **Energy Captured** (Capture) and **Percentage Capture**
261 (%Capture). Energy Captured (in mJ) represents the total energy measured across all the
262 major components of a given layer block or the entire model. Due to the limited precision
263 of the instrumentation, we neglect negligible contributors (e.g., residual connections) and
264 introduce %Capture to indicate how well the methodology accounts for the model’s overall
265 energy usage. Specifically, %Capture is the ratio of the measured Energy Captured to the
266 measured model’s energy consumption, expressed as a percentage:

$$267 \text{Capture} = \sum_{i=1}^N \bar{E}_i, \quad \% \text{Capture} = \frac{\sum_{i=1}^N \bar{E}_i}{\bar{E}_{\text{model}}} \times 100$$

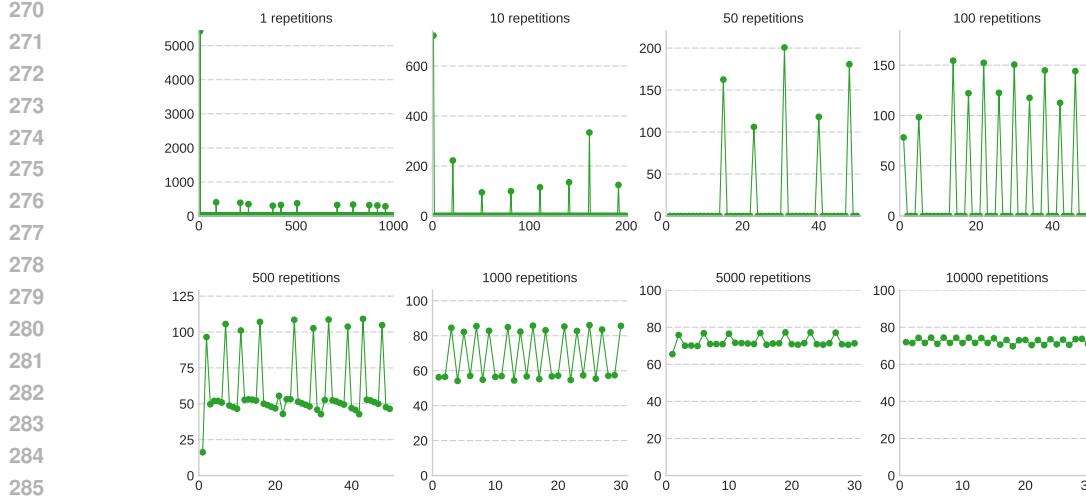


Figure 2: Energy Measurement across multiple trials (T) for varying repetition count N . As N increases, number of reliable readings and precision increases significantly.

4.2.2 COMPUTATIONAL EFFICIENCY

We use FLOPs to quantify the computations executed by each component for a given input. The PyTorch Profiler is employed to measure both the FLOPs and the GPU execution time (in μs), providing insights into the computational efficiency of each component. To assess the energy cost per unit of computation, we define two metrics given as

$$\text{E/FLOP} = \frac{\text{Energy [mJ]}}{\text{FLOPs} \times 10^{-9}}, \quad \Delta E/\Delta \text{FLOP} = \frac{\Delta \text{Energy [mJ]}}{\Delta \text{FLOPs} \times 10^{-9}}.$$

The first metric, E/FLOP (expressed in mJ/GFLOP), characterizes the average energy cost per unit of floating-point computation. Lower values indicate higher energy efficiency, whereas elevated values often signal memory-bound or communication-dominated operations, where energy usage is not directly proportional to computational intensity. The second metric, $\Delta E/\Delta \text{FLOP}$, captures the marginal energy consumed (in mJ) per additional GFLOP and isolates the energy sensitivity of a component to increases in computational demand.

4.3 MODELS

To conduct a systematic study of energy consumption across architectural paradigms, we consider four representative classes of Transformer-based models namely **Encoder-only** models, **Decoder-only** models, **Encoder-Decoder** models and sparse-activated **Mixture of Experts** (MOE) variants. Our model selection aims to balance breadth of architectural diversity with controlled comparisons of scale and design choices.

We evaluate eight widely-used Encoder only models namely BERT-base, BERT-large (Devlin et al., 2019), ALBERT-base, ALBERT-large (Lan et al., 2020), RoBERTa-base, RoBERTa-large (Liu et al., 2019) and distilled variants DistilBERT and DistilBERT (Sanh et al., 2020). Base and large versions allow us to isolate the effect of model size on energy usage where distilled counterparts enable comparison with lightweight compression techniques. To represent contemporary LLMs i.e. Decoder-only models, we experiment with instruction-tuned variants of four key open-source families namely LLaMA 3.2-3B (AI@Meta, 2024), Gemma 3-4B (Team, 2024), Qwen 2.5-3B (Yang et al., 2024), and Phi-4-4B (Abdin et al., 2024). We focus specifically on single-token generation to control for variability in output sequence length and to minimize cache based auto-regressive generation. We also evaluate CLEAR on two well-established sequence-to-sequence models, namely BART (Lewis et al., 2019) and FLAN-T5 (Chung et al., 2022) and a sparse-activated MoE, GPT-OSS-20B (OpenAI, 2025)

Component	8 Tokens		32 Tokens		64 Tokens		96 Tokens		128 Tokens	
	Avg.	Std. Dev.	Avg.	Std. Dev.	Avg.	Std. Dev.	Avg.	Std. Dev.	Avg.	Std. Dev.
GPT-OSS 20B										
Attention Block	53.261	1.677	64.147	0.686	75.161	0.76	93.91	0.779	100.701	1.045
MLP	685.408	12.61	776.905	3.166	867.687	0.867	958.134	1.406	1046.2	1.187
Norm. (All)	9.324	0.729	10.787	0.825	12.702	1.056	13.443	1.422	14.639	1.108
Captured (Block)	747.993	-	851.839	-	955.55	-	1065.487	-	1161.541	-
Block	731.905	12.456	856.309	1.428	951.869	0.805	1057.01	0.881	1157.197	1.181
% Capture (Block)	102.198	-	99.478	-	100.387	-	100.802	-	100.375	-
Embedding Layer	0.568	0.215	0.627	0.282	1.061	0.41	1.077	0.434	0.766	0.357
LM Head	443.391	1.108	452.139	0.988	460.383	0.988	475.22	1.265	483.515	1.242
Final Layer Norm.	4.695	0.368	5.14	0.361	6.071	0.496	6.625	0.525	7.221	0.466
Captured (Model)	18014.38	-	21009.32	-	23312.36	-	25851.15	-	28264.24	-
Model	18447.5	63.784	21366.69	103.479	24126.47	12.67	26634.05	15.33	28801.98	2.867
% Capture (Model)	97.652	-	98.327	-	96.626	-	97.061	-	98.133	-

Table 1: Energy Consumption for GPT-OSS-20B model across different token length with %Capture (96+%) for Block and Full Model and Std. Deviation across 20 trials.

5 RESULTS

RQ1) HOW CONSISTENT ARE ENERGY MEASUREMENTS ACROSS REPEATED TRIALS, AND HOW COMPLETE IS THE CAPTURED ENERGY RELATIVE TO THE TOTAL CONSUMPTION?

Completeness of Energy Captured: Despite the omission of very small and negligible components, the overall %Capture at both the block and model level consistently remains above 90% (Refer Table 1) This demonstrates that summing the measured energies of individual components provides a reliable and near-complete estimate of the total energy consumption dictated by the model’s architecture. However we consistently observe low %Capture(Block) capture of ALBERT variants possible due to factorized embeddings causing higher idle energy consumption.

Consistency Across trials : Using CLEAR, we find that the average standard deviation of the measured component energies consistently remains below 9.5% of the respective mean values for components that consume 5mJ of energy. Furthermore, we observe a clear empirical trend, as both component size and execution time increases, the relative standard deviation decreases. For example, components with energy consumption in the range of 0–5 mJ exhibit around 20+% standard deviation, whereas those consuming 1 J show deviations as low as 0.1 % (Refer Fig 10 in Appendix). Such behavior is expected as shorter execution times result in fewer sensor readings, making the measurements more susceptible to idle-energy noise. Typically, measurements in the 0–5 mJ range are close to the sensor’s own precision limit, causing higher % Std. Deviation with respect to the Average.

RQ2) HOW DOES THE MODEL BEHAVES FROM AN ENERGY PERSPECTIVE UNDER VARYING FLOATING-POINT PRECISIONS?

Across all model architectures, we observe that normalization layers consume more energy in FP16 precision than in FP32 precision, even under identical settings. This stems from the common practice of casting tensors to 32 bit floating point precision for numerical stability during normalization and then converting back, with the type conversions introducing measurable energy overhead. For most encoder-only models, the Attention and Feed-Forward blocks consume roughly comparable amounts of energy (Refer Fig 6 and 7 in Appendix). However as model size grows, the dense FFN layers scale up in parameters, so the Attention block accounts for a smaller share of the total energy, mirroring the pattern seen in large decoder-only models (around 3–4 B parameters). Although upgrading the floating-point precision from 16 to 32 increases the absolute energy consumption, the relative share of energy consumed by the Attention, MLP, and LM-Head components remains virtually unchanged.

RQ3) HOW DOES THE ENERGY/FLOPs RATIO VARY ACROSS INPUT TOKEN LENGTH?

As shown in figure 3, we find that across all input token lengths the Attention mechanism consistently exhibits a higher energy-to-FLOPs ratio than an individual layer or the full model

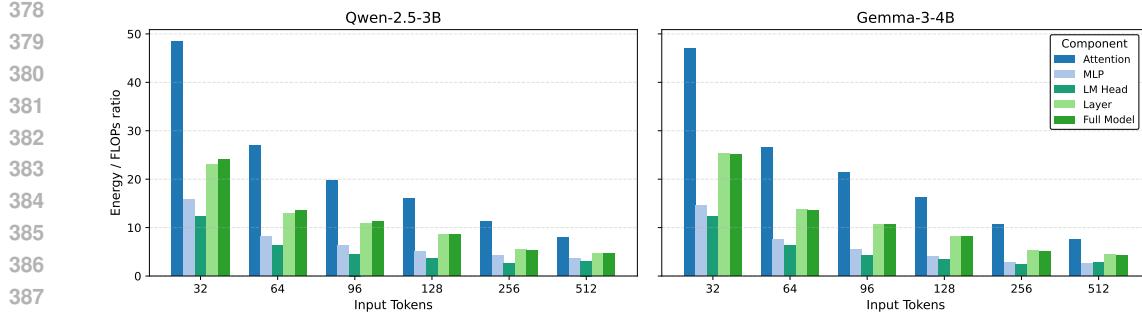


Figure 3: E/FLOPs ratio for model components in Qwen-2.5-3B (left) and Gemma-3-4B (right). Attention consistently shows highest E/FLOPs ratio across all input lengths.

as a whole. In contrast, the MLP and LM-head layers consume noticeably less energy per FLOP, which means they perform more computations for every joule of energy spent. This highlights that, from an energy-efficiency perspective, Attention is the most computationally expensive sub-component. This inefficiency stems from the nature of attention computations. Unlike the dense matrix multiplications in the MLP and LM head, which are regular, highly parallelizable operations that GPUs are specifically optimized to accelerate whereas attention involves query–key dot products, scaling, softmax operations, and complex memory access patterns. These steps introduce memory traffic and synchronization overheads, causing the hardware to spend proportionally more energy per unit of floating-point computation.

As shown in figure 4 and 9 (Appendix), the energy to FLOPs ratio for every individual component steadily decreases as the input sequence length grows i.e. when we feed the model longer input sequences, each FLOP consumes lesser energy, not only for the complete model, but also for the Attention, MLP, and LM-head blocks. This trend reflects that the fixed costs of computation and memory movement are amortized over more tokens. The per-token energy overheads are diluted and the compute resources are utilized more effectively.

RQ4) IS NUMBER OF FLOATING-POINT OPERATIONS (FLOPs) A RELIABLE INDICATOR OF A COMPONENT’S ENERGY CONSUMPTION?

As shown in Fig. 4, the energy-to-FLOPs ratio (E/FLOPs) decreases consistently as the input sequence length grows. As discussed in **RQ3**, the trend points to the presence of a fixed energy overhead, that is a baseline energy cost E_0 likely independent of the number of floating-point operations. Such a cost likely stems from non-scaling aspects of the computation, including memory movements, cache initialization, and kernel-launch overheads, all of which incur an essentially constant energy expenditure regardless of token count.

To quantify the marginal energy required per unit of additional computation, we examined the incremental energy cost per FLOP, $\Delta E / \Delta FLOPs$. Unlike the absolute ratio $E/FLOPs$, the marginal quantity remains approximately constant as the input sequence length increases across all model components. This stability provides strong empirical evidence that FLOPs are indeed the main driver of the **variable** portion of energy consumption. In other words, while total energy can be decomposed as

$$E(L) \approx E_0 + k \cdot \text{FLOPs}(L) \quad (5)$$

with L denoting input length and k a constant of proportionality, only the second term grows nearly linearly with the computational workload (FLOPs).

Notably, the marginal coefficient k is, however, component-dependent. It is noticeably higher for the Attention mechanism than for other subsystems. This can be attributed to the fact that Attention performs repeated key–value cache updates and experiences substantial memory–compute synchronization overheads, both of which increase the effective energy cost per FLOP.

Overall, the empirical observations indicate that energy consumption of different components can be well approximated with two term model comprising of a fixed overhead and a FLOP

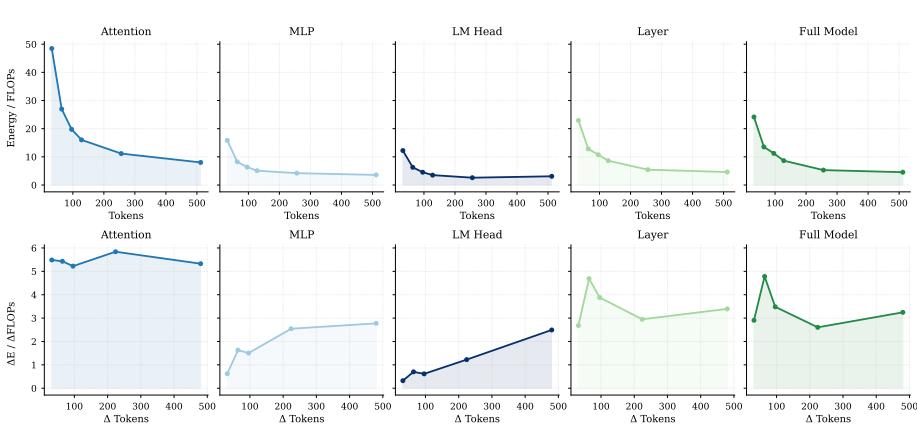


Figure 4: E/FLOPs (top row) and $\Delta E/\Delta \text{FLOPs}$ (bottom row) ratios for components of Qwen-2.5-3B model across Tokens and Δ Tokens respectively. E/FLOP ratio decreases on increasing input token length but marginal energy consumed per FLOP remains nearly constant.

proportional component. This indicates that, the idea of measuring total energy consumed by a model and then distributing it proportionally across components solely based on FLOPs is overly simplistic. Both the existence of the fixed overhead and the component-specific marginal cost k must be accounted for to estimate energy usage of each component accurately.

6 DISCUSSION

Most sustainability studies have primarily focused on model-level or system-level energy consumption treating LLMs as *monolithic entities* and only a little attention is paid to the heterogeneity of its internal components from a sustainability perspective. CLEAR’s contribution to measure energy consumption at component-level granularity has direct implications for research community as it provides a systematic methodology to reliably measure internal energy dynamics and enable targeted energy optimizations at the design level of model architecture.

As discussed in **RQ3** and **RQ4**, we observe that each component consumes energy disproportionately, posing a threat to use FLOPs and related metrics as convenient proxies energy consumption (Getzner et al., 2023; Özcan et al., 2025) as component-level disparities are systematically obscured by model-level aggregate measurements. Appendix C for Limitations

As seen in **RQ1**, CLEAR demonstrates statistically reliable and sufficiently complete component-wise energy profiling that can be employed to support comparative studies and draw robust conclusions about energy implications of specific design choices, rather than relying on estimates. CLEAR establishes a foundation for future work on predictive modeling so that the energy costs can be computed based on architectural design choices like hidden dimensions, number of layers, etc. allowing accurate, generalizable prediction of component-wise energy dynamics in the early design stages. This aligns with the growing emphasis on Green AI and the need for energy-aware, sustainable AI system design. (Bolón-Canedo et al., 2024; Sánchez-Mompó et al., 2025b; Rózycki et al., 2025)

Taken together, these findings underscore that sustainability in AI must be treated as a **first-class research objective rather than an afterthought**. By moving beyond aggregate model-level reporting to examine component-level dynamics, we aim to motivate the software and AI research communities to pursue progress that is both holistic and environmentally responsible, driven by a proactive rather than reactive mindset. Looking forward, we hope this work inspires future research to integrate energy considerations into every stage of model development, fostering AI systems that are not only performant but also sustainable.

486 **7 REPRODUCIBILITY STATEMENT**
 487

488 To ensure transparency and reproducibility, we are committed to making our research
 489 accessible. We provide comprehensive experimental details in the paper, and all code will
 490 be publicly released upon publication. All experiments were conducted using open-source
 491 LLMs.

492
 493 **8 ETHICS STATEMENT**
 494

495 This work presents a methodology for measuring energy consumption at the component level
 496 in transformer architectures, a dimension that has not been systematically studied so far.
 497 Using this methodology, we analyze how energy is distributed across different components
 498 of a transformer. Our findings are intended to advance the development of more efficient
 499 transformer architectures and reliable methods for energy estimation. Refer D for LLM
 500 Usage statement

501
 502 **REFERENCES**
 503

504 Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar,
 505 Michael Harrison, Russell J. Hewett, Mojan Javaheripi, Piero Kauffmann, James R. Lee,
 506 Yin Tat Lee, Yuanzhi Li, Weishung Liu, Caio C. T. Mendes, Anh Nguyen, Eric Price,
 507 Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Xin Wang, Rachel Ward,
 508 Yue Wu, Dingli Yu, Cyril Zhang, and Yi Zhang. Phi-4 technical report, 2024. URL
 509 <https://arxiv.org/abs/2412.08905>.

510 AceCloud. Ai training vs inference: Key differences. <https://acecloud.ai/blog/ai-training-vs-inference/>, 2024.

513 AI@Meta. Llama 3 model card. 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

516 Negar Alizadeh and Fernando Castor. Green ai: a preliminary empirical study on energy
 517 consumption in dl models across different runtime infrastructures. In *Proceedings of the*
 518 *IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for*
 519 *AI, CAIN 2024*, page 134–139. ACM, April 2024. doi: 10.1145/3644815.3644967. URL
 520 <http://dx.doi.org/10.1145/3644815.3644967>.

521 Anonymous. Part-time power measurements: nvidia-smi’s lack of responsiveness for short
 522 workloads. arXiv preprint arXiv:2312.02741, 2023. URL <https://arxiv.org/abs/2312.02741>. Finds NVML power sampling typically occurs at 20–50 Hz, with averaged values
 523 over 1 s.

526 Lasse F. Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. Carbontracker:
 527 Tracking and predicting the carbon footprint of training deep learning models, 2020. URL
 528 <https://arxiv.org/abs/2007.03051>.

530 Verónica Bolón-Canedo, Laura Morán-Fernández, Brais Cancela, and Amparo Alonso-
 531 Betanzos. A review of green artificial intelligence: Towards a more sustainable future.
 532 *Neurocomput.*, 599(C), September 2024. ISSN 0925-2312. doi: 10.1016/j.neucom.2024.
 533 128096. URL <https://doi.org/10.1016/j.neucom.2024.128096>.

534 Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan
 535 Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane
 536 Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros,
 537 Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu,
 538 Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff
 539 Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling
 instruction-finetuned language models, 2022. URL <https://arxiv.org/abs/2210.11416>.

- 540 Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of
 541 deep bidirectional transformers for language understanding, 2019. URL <https://arxiv.org/abs/1810.04805>.
- 543
- 544 Earth.org. The environmental impact of chatgpt. <https://earth.org/environmental-impact-chatgpt/>, 2023.
- 545
- 546 EcoTree. How much co2 does a tree absorb? <https://ecotree.green/en/how-much-co2-does-a-tree-absorb>, 2023. Accessed: 2025-09-05.
- 547
- 548
- 549 Ahmad Faiz, Sotaro Kaneda, Ruhan Wang, Rita Osi, Prateek Sharma, Fan Chen, and Lei
 550 Jiang. Llmcarbon: Modeling the end-to-end carbon footprint of large language models,
 551 2024. URL <https://arxiv.org/abs/2309.14393>.
- 552
- 553 Jared Fernandez, Clara Na, Vashisth Tiwari, Yonatan Bisk, Sasha Luccioni, and Emma
 554 Strubell. Energy considerations of large language model inference and efficiency optimiza-
 555 tions, 2025. URL <https://arxiv.org/abs/2504.17674>.
- 556
- 557 Johannes Getzner, Bertrand Charpentier, and Stephan Günemann. Accuracy is not the
 558 only metric that matters: Estimating the energy consumption of deep learning models,
 559 2023. URL <https://arxiv.org/abs/2304.00897>.
- 560
- 561 Google. Environmental report 2025. <https://sustainabilitymag.com/articles/google-environmental-report-2025>, 2025.
- 562
- 563 Nidhal Jegham, Marwan Abdelatti, Lassad Elmoubarki, and Abdeltawab Hendawi. How
 564 hungry is ai? benchmarking energy, water, and carbon footprint of llm inference, 2025.
 565 URL <https://arxiv.org/abs/2505.09598>.
- 566
- 567 Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and
 568 Radu Soricut. Albert: A lite bert for self-supervised learning of language representations,
 569 2020. URL <https://arxiv.org/abs/1909.11942>.
- 570
- 571 Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed,
 572 Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: denoising sequence-to-
 573 sequence pre-training for natural language generation, translation, and comprehension.
CoRR, abs/1910.13461, 2019. URL <http://arxiv.org/abs/1910.13461>.
- 574
- 575 Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy,
 576 Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert
 577 pretraining approach, 2019. URL <https://arxiv.org/abs/1907.11692>.
- 578
- 579 Alexandra Sasha Luccioni, Sylvain Viguer, and Anne-Laure Ligozat. Estimating the carbon
 580 footprint of bloom, a 176b parameter language model. *Journal of Machine Learning
 Research*, 24(253):1–15, 2023. URL <http://jmlr.org/papers/v24/23-0069.html>.
- 581
- 582 Sasha Luccioni and collaborators. Announcing AI Energy Score Ratings. 2025. Accessed:
 583 2025-09-06.
- 584
- 585 CBS News. Ai's growing environmental impact: Study
 586 on google gemini. <https://www.cbsnews.com/news/ai-environment-impact-study-energy-usage-google-gemini-prompt/>, 2025.
- 587
- 588 NVIDIA. Nvidia rtx 6000 ada generation datasheet. Technical Datasheet (PDF), 2023. URL
 589 <https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/rtx-6000/proviz-print-rtx6000-datasheet-web-2504660.pdf>. Official specifications
 590 for the RTX 6000 Ada, including CUDA/RT/Tensor cores, memory, bandwidth, and
 591 power.
- 592
- 593 OpenAI. gpt-oss-120b gpt-oss-20b model card, 2025. URL <https://arxiv.org/abs/2508.10925>.

- 594 Alyssa Pinnock, Shakya Jayakody, Kawsher A Roxy, and Md Rubel Ahmed. Edgeprofiler: A
 595 fast profiling framework for lightweight llms on edge using analytical model, 2025. URL
 596 <https://arxiv.org/abs/2506.09061>.
- 597
- 598 PNY. Nvidia rtx 5000 ada generation datasheet. Product Datasheet (PDF),
 599 2023. URL <https://www.pny.com/en-eu/File%20Library/Professional/DATASHEET/WORKSTATION/PNY-NVIDIA-RTX-5000-Ada-Generation-Datasheet.pdf>. Specifications
 600 include CUDA cores, Tensor/RT cores, memory capacity, bandwidth, and board power.
- 601
- 602 Saurabh Singh Rajput, Tim Widmayer, Ziyuan Shang, Maria Kechagia, Federica Sarro, and
 603 Tushar Sharma. Enhancing energy-awareness in deep learning through fine-grained energy
 604 measurement, 2024. URL <https://arxiv.org/abs/2308.12264>.
- 605
- 606 Rafał Różycki, Dorota Agnieszka Solarska, and Grzegorz Waligóra. Energy-aware machine
 607 learning models—a review of recent techniques and perspectives. *Energies*, 18(11), 2025.
 608 ISSN 1996-1073. doi: 10.3390/en18112810. URL <https://www.mdpi.com/1996-1073/18/11/2810>.
- 609
- 610 Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled
 611 version of bert: smaller, faster, cheaper and lighter, 2020. URL <https://arxiv.org/abs/1910.01108>.
- 612
- 613 Adrián Sánchez-Mompó, Ioannis Mavromatis, Peizheng Li, Konstantinos Katsaros, and Aftab
 614 Khan. Green mlops to green genops: An empirical study of energy consumption in discriminative
 615 and generative ai operations. *Information*, 16(4):281, March 2025a. ISSN 2078-2489.
 616 doi: 10.3390/info16040281. URL <http://dx.doi.org/10.3390/info16040281>.
- 617
- 618 Adrián Sánchez-Mompó, Ioannis Mavromatis, Peizheng Li, Konstantinos Katsaros, and Aftab Khan. Green mlops to green genops: An empirical study of energy consumption in discriminative
 619 and generative ai operations. *Information*, 16(4), 2025b. ISSN 2078-2489.
 620 doi: 10.3390/info16040281. URL <https://www.mdpi.com/2078-2489/16/4/281>.
- 621
- 622 Gemma Team. Gemma: Open models based on gemini research and technology, 2024. URL
 623 <https://arxiv.org/abs/2403.08295>.
- 624
- 625 TechTarget. Ai inference vs training: Key differences and trade-offs.
 626 <https://www.techtarget.com/searchenterpriseai/tip/AI-inference-vs-training-Key-differences-and-tradeoffs>, 2025.
- 627
- 628 U.S. Environmental Protection Agency. Greenhouse gas emissions from
 629 a typical passenger vehicle. <https://www.epa.gov/greenvehicles/greenhouse-gas-emissions-typical-passenger-vehicle>, 2023. Accessed: 2025-09-05.
- 630
- 631
- 632 An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li,
 633 Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong
 634 Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang,
 635 Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin
 636 Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui
 637 Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li,
 638 Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang,
 639 Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan,
 640 Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. Qwen2
 641 technical report, 2024. URL <https://arxiv.org/abs/2407.10671>.
- 642
- 643 Miray Özcan, Philipp Wiesner, Philipp Weiß, and Odej Kao. Quantifying the energy
 644 consumption and carbon emissions of llm inference via simulations, 2025. URL <https://arxiv.org/abs/2507.11417>.
- 645
- 646
- 647

648 APPENDIX
649650 A TRANSFORMER ARCHITECTURES
651

652 Despite architectural differences, the transformer based models share a set of common
653 computational primitives. The detailed flow of how the model produces its output is
654 illustrated in the following.
655

656 Let \mathcal{V} denote the vocabulary space, and $x = (x_1, \dots, x_T)$, $x_t \in \mathcal{V}$ be an input token sequence
657 of length T . A tokenizer $\mathcal{T} : \mathcal{V} \rightarrow \{1, \dots, |\mathcal{V}|\}$ maps tokens to discrete indices. The indices
658 are embedded into a continuous space by the **Embedding Layer** $E \in \mathbb{R}^{|\mathcal{V}| \times d}$:

$$659 \quad h_t^0 = E[x_t] + P_t, \quad t = 1, \dots, T,$$

660 where $P_t \in \mathbb{R}^d$ denotes the positional embedding and d is the hidden dimensionality. Embed-
661 dings are then passed to a set of Transformer layers each consisting of an **Attention** (Attn)
662 block followed by a position-wise Feed-Forward **MLP** block, interleaved with residual con-
663 nections and **Normalization** blocks for numerical stability, as depicted in the Transformer
664 block of Figure 1. For layer $\ell \in \{1, \dots, L\}$, the computations are:
665

$$666 \quad \tilde{h}^\ell = \text{LN}(h^{\ell-1}), \quad z^\ell = h^{\ell-1} + \text{Attn}(\tilde{h}^\ell), \quad h^\ell = z^\ell + \text{FFN}(\text{LN}(z^\ell)).$$

667 However, the placement of Normalization blocks vary across different models and can be
668 applied at different stages within a layer block. Post all layers, the final hidden state
669 $h^L \in \mathbb{R}^{T \times d}$ is projected onto the vocabulary using the language modeling head:
670

$$671 \quad \hat{y}_t = \text{softmax}(h_t^L W_{\text{LM}}^\top), \quad W_{\text{LM}} \in \mathbb{R}^{|\mathcal{V}| \times d}.$$

673 A.1 ENCODER-ONLY
674

675 Encoder-only architectures (e.g., BERT, RoBERTa, ALBERT) compute contextualized token
676 representations using bidirectional self-attention. They are commonly used for classification,
677 token-level prediction and masked-language-modeling objectives. The encoder processes
678 the full input x in parallel, producing $H^L = (h_1^L, \dots, h_T^L) \in \mathbb{R}^{T \times d}$ which can be pooled or
679 projected to task heads.
680

Encoder-only architectures typically use a *dense attention pattern*, meaning that no causal
mask is applied to restrict attention. Formally, the attention operation is defined as

$$682 \quad \text{Attn}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V,$$

683 where every token can attend to every other token in the sequence. This design produces
684 *symmetric contextualization*, since information can flow bidirectionally across tokens. However,
685 it comes with a computational cost of $\mathcal{O}(T^2d)$ per layer (due to the pairwise interactions
686 between all tokens) and a memory cost of $\mathcal{O}(T^2)$ for storing the attention weights.
687

Different tasks attach specialized output heads on top of the final hidden states $h_t^L \in \mathbb{R}^d$.
For **token-level classification**, each token representation is projected into the label space
using a weight matrix W_{tok} . The predicted class distribution for token t is given by

$$692 \quad \hat{y}_t = \text{softmax}(h_t^L W_{\text{tok}}^\top).$$

693 For **sequence-level classification**, the hidden state of a special token such as [CLS] serves
694 as a summary vector for the entire sequence. This representation h_{cls}^L is then passed through
695 a classifier, typically implemented as a multi-layer perceptron (MLP):
696

$$697 \quad h_{\text{cls}}^L \mapsto \text{MLP}(h_{\text{cls}}^L).$$

698 For **masked language modeling (MLM)**, the prediction head reuses the input embedding
699 matrix E to tie input and output representations. In this case, the output weight matrix is
700 defined as
701

$$W_{\text{LM}} = E.$$

This weight sharing enforces consistency between how tokens are encoded and how they are predicted.

Encoder-only designs maximize parallelizability during training as the whole sequence is processed concurrently, but the T^2 attention cost and the need to store full-layer activations drive both memory bandwidth and energy cost during training.

Model	# of Layers	Hidden Dimension	Attention Heads	Feed-Forward Dimension	Parameters	Special Features
google-bert/bert-base-uncased	12	768	12	3072	110M	Uses [CLS] token for classification
google-bert/bert-large-uncased	24	1024	16	4096	340M	Larger variant of BERT with higher representational capacity
albert/albert-base-v2	12	768	12	3072	12M	Parameter-sharing across layers and factorized embedding
albert/albert-large-v2	24	1024	16	4096	18M	Deeper network with the same parameter-sharing strategy
distilbert/distilbert-base-uncased	6	768	12	3072	66M	Distilled version of BERT with 40% fewer parameters
distilroberta/distilroberta-base	6	768	12	3072	82M	Distilled version of RoBERTa retaining most performance
FacebookAI/roberta-base	12	768	12	3072	125M	Improved pretraining and dynamic masking
FacebookAI/roberta-large	24	1024	16	4096	355M	Larger RoBERTa model with improved pretraining

Table 2: Architectural Details of Encoder-Only Models

A.2 DECODER-ONLY (AUTOREGRESSIVE)

Decoder-only (autoregressive) models (e.g. Llama3.1, GPT) perform next-token prediction and are optimized for generative tasks. The decoder processes tokens causally with a triangular mask in which each position can attend only to tokens at previous positions (and itself) to enforce autoregressive factorization. The causal mask M_{causal} has entries 0 for allowed positions and $-\infty$ for disallowed future positions, implementing the triangular attention.

Inference optimization: KV-caching. During autoregressive generation, previously computed keys and values can be cached: for step t only the new query interacts with stored $K_{1:t-1}, V_{1:t-1}$. This reduces per-step attention cost from $\mathcal{O}(t^2d)$ to $\mathcal{O}(td)$ (amortized), and reduces the energy per generated token substantially at inference time. Training a decoder-only model still incurs full-sequence $\mathcal{O}(T^2d)$ attention cost, but inference benefits from KV-caching. Energy per generated token at inference depends on cache memory bandwidth and per-layer dot-product costs; thus memory movement for KV-cache and tiled attention matmuls can dominate measured energy.

The set of activation hooks \mathcal{A}_{dec} defined similarly to the encoder case, but is adapted to handle causal inputs and cached key-value states and it stores the intermediate activations $\alpha_{\ell,t}^{\text{attn_in}}$ and $\alpha_{\ell,t}^{\text{kv}}$ for each layer ℓ and time step t . For profiling isolated attention at generation-time, replaying using cached KV tensors models the exact inference cost. For the purpose of our analysis, we primarily consider the energy and computation associated with the generation of a **single token**, where KV-caching is not utilized.

Decoder-only models are majorly different from Encoder-only models as they prioritize autoregressive causality in which causal masking changes attention sparsity and reduces parallelism during sequence generation. KV-caching and rotary/relative positional encodings are often used to support long-context amortized inference and decoder-only models commonly use tied input/output embeddings to reduce parameter counts, and favor pre-norm residual stacks for stability in deep networks.

A.3 ENCODER-DECODER

Encoder-Decoder models, also known as sequence-to-sequence (Seq2Seq) architectures, are widely used for tasks requiring input-to-output transformations such as machine translation, summarization, and code generation. Formally, given an input sequence $x = (x_1, \dots, x_{T_{\text{in}}})$,

Architecture Detail	Qwen2.5-3B-Instruct	Phi-4-Mini-Instruct	Llama-3.2-3B-Instruct	Gemma-3-4B-IT
Parameters	3.09B total (2.77B non-embedding)	4B	3.21B	4B
Layers	36	32	28	34
Hidden Size / Head Dim	2048 hidden, 128 per head	3072 hidden, 128 per head	3072 hidden, 128 per head	2560 hidden, 128 per head
Attention Structure	GQA: 16 query heads, 2 KV heads; RoPE; QKV bias; output proj. biasless	GQA: 24 query heads, 8 KV heads; Fractional RoPE (25% position-agnostic); KV cache optimized	GQA: 24 query heads, 8 KV heads; RoPE; no bias in projections	Local+Global attention mix; Q-proj: 2048-d, K/V-proj: 1024-d; q.norm, k.norm applied
MLP / FFN Dimension	SwiGLU, 11008 (up+gate), 2048 down	SiLU, 16384 (gate+up), 8192 down	SiLU, 8192 up, 3072 down	GELU-Tanh, 10240 up, 2560 down
Normalization	RMSNorm, $\epsilon = 1e-6$, applied input + post-attn	RMSNorm, $\epsilon = 1e-5$, input + post-attn	RMSNorm, $\epsilon = 1e-5$, input + post-attn	RMSNorm, $\epsilon = 1e-6$, input + post-attn + pre/post-FFN
Embeddings	151,936 vocab, 2048-d, tied in/out	200,064 vocab, 3072-d, tied in/out (padding idx=199999)	128,256 vocab, 3072-d, tied in/out	262,208 vocab, 2560-d, tied in/out, scaled embeddings
Context Length	32K tokens (gen up to 8K)	Long-context via KV optimization, tested up to $\sim 128K$	128K tokens, efficient GQA	128K tokens; local layers span 1K, 1 global layer every 5 locals
Special Features	RoPE, SwiGLU, QKV bias, high multilingual coverage	GQA w/ reduced KV cache, fractional RoPE, tuned LR schedule	Optimized transformer, SFT+RLHF alignment, multilingual	Local-global hybrid attention, multimodal (SigLIP image encoder), Pan & Scan for variable resolution

Table 3: Detailed architectural comparison of Decoder-Only Qwen2.5-3B, Phi-4-Mini, Llama-3.2-3B, and Gemma-3-4B instruction-tuned models.

the encoder maps it to a sequence of hidden representations $H = (h_1, \dots, h_{T_{\text{in}}})$, and the decoder generates an output sequence $y = (y_1, \dots, y_{T_{\text{out}}})$ autoregressively, conditioned on H .

The encoder is a stack of L_e Transformer layers that performs contextual embedding of the input tokens. Each layer typically consists of:

- An **Attention** (Attn) mechanism that captures global dependencies within the input sequence i.e. for layer ℓ
- $$\tilde{h}^{\ell-1} = \text{LN}(h^{\ell-1}), \quad h^\ell = h^{\ell-1} + \text{Attn}^\ell(\tilde{h}^{\ell-1}), \quad h^\ell = h'^\ell + \text{FFN}^\ell(\text{LN}(h'^\ell)).$$
- **Feedforward Network** that adds per-position nonlinear transformation to learn deeper features.

The encoder produces rich representations that capture semantic and syntactic relationships within the input sequence.

The decoder is also a stack of L_d Transformer layers, each consisting of:

- **Masked self-attention** which ensures autoregressive generation by attending only to previous positions.
- **Encoder-decoder cross-attention** mechanism to attend to the encoder hidden states H , incorporating information from the entire input sequence into each decoding step.
- A **Feedforward network** similar to the encoder.

Mathematically, for decoder layer ℓ :

$$\begin{aligned} \tilde{u}^{\ell-1} &= \text{LN}(u^{\ell-1}), \quad s_{\text{self}}^\ell = \text{Attn}_{\text{causal}}(\tilde{u}^{\ell-1}), \\ s_{\text{cross}}^\ell &= \text{Attn}_{\text{cross}}(\text{LN}(u^{\ell-1} + s_{\text{self}}^\ell), H), \quad u^\ell = \text{FFN}(\text{LN}(u^{\ell-1} + s_{\text{self}}^\ell + s_{\text{cross}}^\ell)). \end{aligned}$$

Compared to encoder-only models, the encoder-decoder architecture introduces a separate decoder stack with cross-attention, which enables output generation conditioned on the full input sequence. In contrast, encoder-only models produce fixed-length or token-level representations that are typically used for classification or embedding tasks, without any autoregressive generation.

When compared to decoder-only models, encoder-decoder architectures separate the input encoding from the output generation, whereas decoder-only models combine both within a single autoregressive stack. This separation allows the encoder to process the entire input sequence in parallel, improving training efficiency. Furthermore, in terms of residual and attention patterns, encoder-decoder models incorporate both self-attention in the decoder and cross-attention between the decoder and encoder outputs, whereas encoder-only and decoder-only architectures contain only a single attention mechanism.

Energy Perspective: The two-stack design of encoder-decoder models increases the total parameter count and memory footprint, resulting in higher energy consumption during training compared to encoder-only models for sequences of the same length. However, the input encoding phase can be fully parallelized across positions, and autoregressive decoder computation can benefit from caching mechanisms during inference, which partially reduces the per-token energy cost

A.4 MIXTURE OF EXPERTS (MoE)

Mixture-of-Experts (MoE) architectures extend standard Transformers by introducing conditional computation i.e. instead of activating all parameters for every input token, only a subset of "expert" networks is selected dynamically. This allows scaling model capacity substantially while keeping per-token computation and energy consumption manageable.

An MoE layer contains E independent feedforward networks, or Experts, each with parameters $\theta_1, \dots, \theta_E$. For a given token representation $h \in \mathbb{R}^d$, the computation is routed through a small subset of $k < E$ experts, typically $k = 2$ or 3

$$m^{\text{MoE}}(h) = \sum_{i \in \text{Top-}k} g_i(h) \text{FFN}_i(h),$$

where $g_i(h)$ is the gating weight assigned by the Router. By activating only a few experts per token, the effective FLOPs per token can be reduced from $\mathcal{O}(E \cdot d \cdot d_{\text{ff}})$ to $\mathcal{O}(k \cdot d \cdot d_{\text{ff}})$.

The Router is a lightweight module that predicts which experts should process a given token:

$$g(h) = \text{softmax}(hW_r), \quad g \in \mathbb{R}^E.$$

It selects the top- k experts according to the largest g_i values. The Router can also include auxiliary losses, such as load-balancing or importance losses, to encourage uniform expert utilization and avoid stragglers, which would increase memory or energy spikes.

By increasing the total number of experts E without increasing k , it is possible to scale the model's representational capacity while incurring only a small incremental energy cost per token. In practice, expert computations for different tokens are often batched across devices, but load imbalance can increase memory movement and create temporary energy spikes due to which careful load-balancing and token assignment become necessary to maintain efficiency.

B HARDWARE SPECIFICATION

The experiments in this paper were carried out using NVIDIA's Ada-Lovelace architecture GPUs, namely the RTX 5000 Ada and RTX 6000 Ada, in order to assess compute and energy performance. The Ada Lovelace architecture is fabricated on a custom 4 nm TSMC process and includes third-generation RT cores and fourth-generation Tensor cores, enabling mixed precision operations (including FP8 with sparsity) that are integral to efficient transformer inference NVIDIA (2023). According to the official datasheets, the RTX 5000 Ada has 12,800 CUDA cores, 100 RT cores, 400 Tensor cores, 32 GB of ECC GDDR6 memory over a 256-bit interface (providing \sim 576 GB/s bandwidth), and a total board power of approximately 250 W PNY (2023). The RTX 6000 Ada model offers 18,176 CUDA cores, 142 RT cores, 568 Tensor cores, 48 GB of ECC GDDR6 memory on a 384-bit interface (\sim 960 GB/s bandwidth), and has a board power of around 300 W NVIDIA (2023). These hardware choices directly

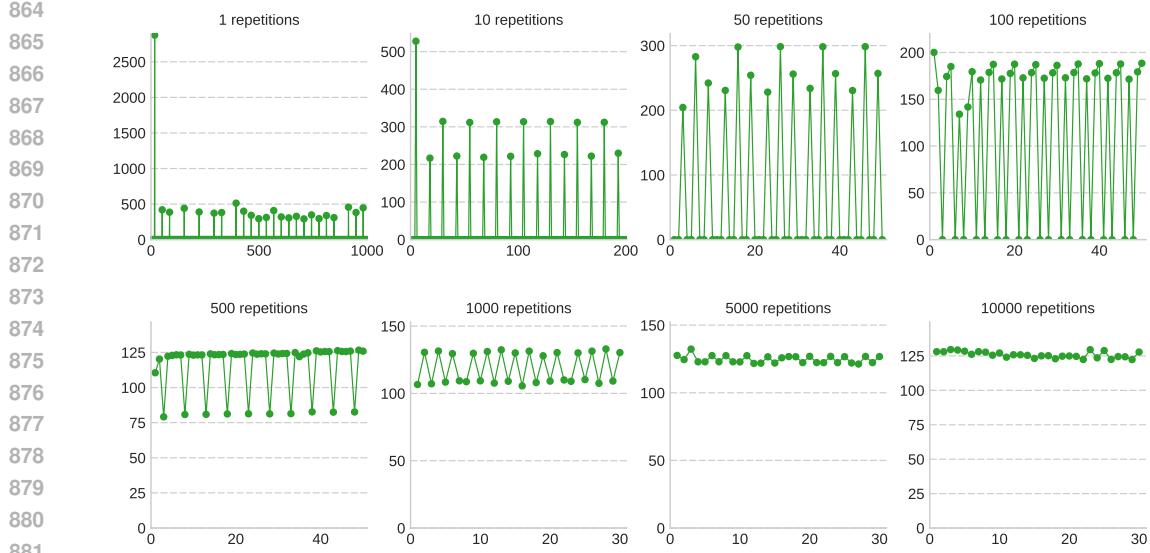


Figure 5: Repetitions performed to eliminate noise in measurement

influence both the sustained compute throughput and the energy-per-FLOP metrics reported in our results.

NVIDIA does not publish the precise NVML power sampling interval for the RTX 5000 Ada or RTX 6000 Ada. Prior work has shown that on modern NVIDIA GPUs, NVML’s power readouts are typically updated at a frequency of 20–50 Hz (i.e., every 20–50 ms), which constrains the granularity of fine-grained energy attribution Anonymous (2023).

C LIMITATIONS

While our study provides the first component-level view of energy consumption in Transformer architectures, a few aspects merit further exploration. First, our energy estimates rely on NVIDIA’s NVML interface and FLOP counts obtained via the PyTorch Profiler. These are well-established tools, but like all measurement frameworks they may carry some inherent uncertainties that may slightly affect the results by negligible margin.

Second, GPU hardware introduces additional variability. Different GPU families and generations apply their own low-level optimization, potentially affecting the energy profiles of specific computational components. Extending this analysis across a wider range of hardware would sharpen our understanding of how these optimizations influence component-wise energy usage. While this is left for future work, to the best of our knowledge, our study remains the first systematic investigation of component-level energy dynamics in Transformer models, providing a foundational understanding of the challenges and opportunities present on current hardware.

Finally, because prior literature offers little precedent for fine-grained energy measurement of individual Transformer components, our validation protocol represents an initial step. Future work can strengthen and expand these validation methods as the research community develops more sophisticated benchmarks and measurement standards.

D LLM USAGE

We used LLMs as writing assistants to help polish and clarify the text in this paper motivated to improve clarity and consistency.

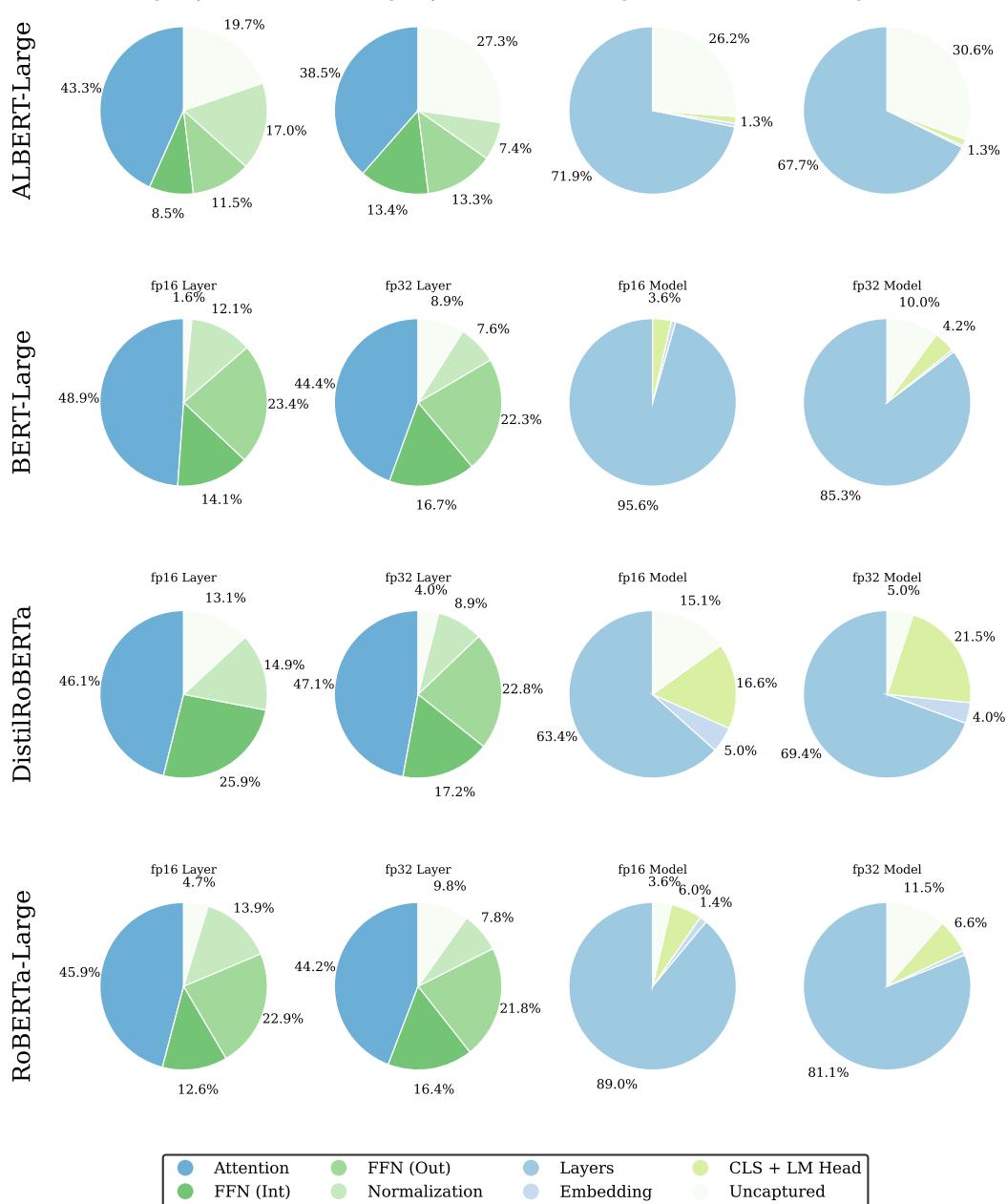


Figure 6: Energy breakdown in large variants of encoder-only models. The first two charts illustrate block-level distributions, while the latter two present distributions across the entire model. FP-16 and FP-32 precisions for each model are shown.

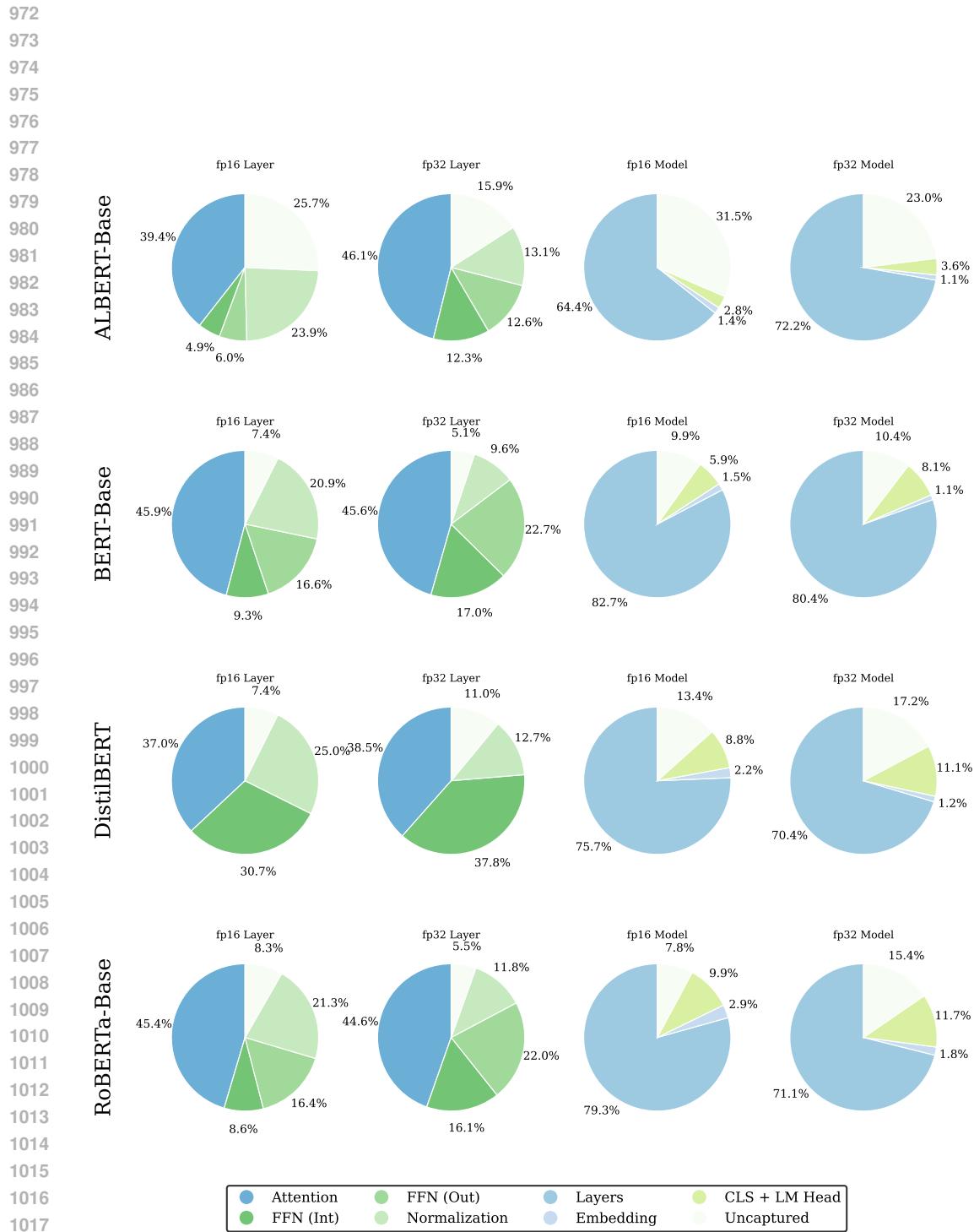


Figure 7: Energy breakdown in encoder-only models. The first two charts illustrate block-level distributions, while the latter two present distributions across the entire model. FP-16 and FP-32 precisions for each model are shown.

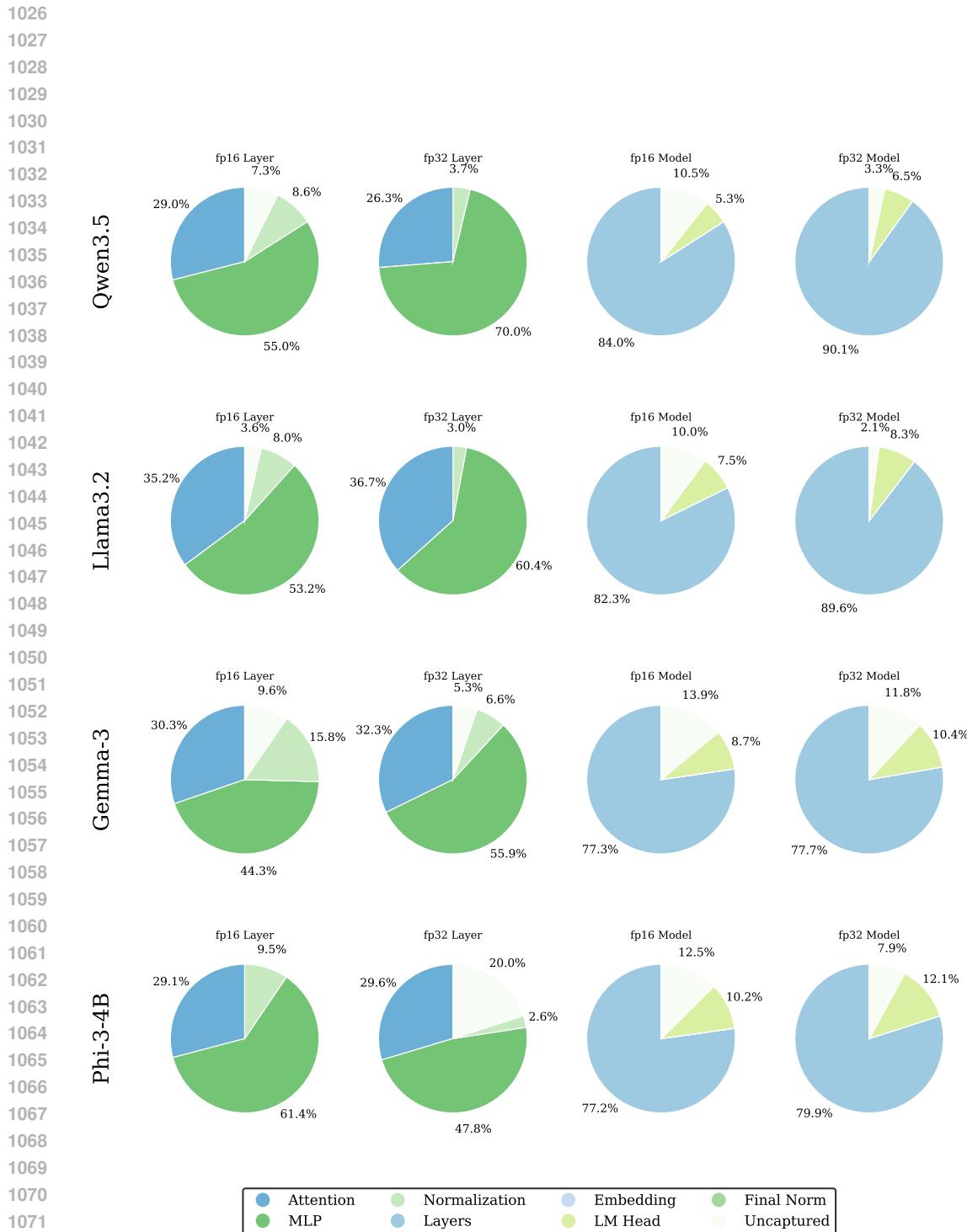


Figure 8: Energy breakdown in decoder-only models. The first two charts illustrate block-level distributions, while the latter two present distributions across the entire model. FP-16 and FP-32 precisions for each model are shown.

Components	Qwen2.5-3B fp32		Qwen2.5-3B fp16		Llama-2.3B fp32		Llama-2.3B fp16		Gemma-4B fp32		Gemma-4B fp16	
	Mean	Std. Dev	Mean	Std. Dev	Mean	Std. Dev						
MLP	113.71	6.687	48.50	2.13	127.24	1.94	54.55	2.01	129.47	1.86	60.07	2.13
Attention	27.64	1.79	33.99	5.96	58.31	3.01	23.88	1.42	42.82	2.11	36.84	1.85
Input Layer Norm	2.59	0.24	6.42	1.02	3.20	0.25	3.61	0.41	3.74	0.45	4.42	0.43
Attention Layer Norm	2.81	0.33	6.80	0.97	3.20	0.32	3.85	0.41	3.49	0.33	4.52	0.36
Block	150.03	3.39	96.19	5.15	192.26	24.92	89.85	1.98	187.54	3.91	124.63	4.54
Capture (Block)	146.75	-	95.71	-	191.95	-	85.89	-	186.54	-	114.83	-
Final Layer Norm	2.96	0.41	6.94	0.97	3.20	0.27	4.20	0.58	3.50	0.33	4.53	0.32
Embedding	0.81	0.26	0.74	0.06	0.69	0.23	0.68	0.24	1.52	0.29	1.28	0.27
LLM Head	459.66	2.18	214.29	3.56	602.02	2.97	274.92	2.96	1040.63	7.25	470.64	8.72
Overall	5995.27	26.77	3685.95	29.11	6029.32	10.68	3261.50	30.99	8086.96	25.25	5248.99	89.34
Capture (Model)	6142.43	-	3962.34	-	6170.53	-	2980.14	-	7953.30	-	5237.99	-

Table 4: Comparison of FP32 and FP16 performance across Qwen2.5-3B, Llama-2.3B, and Gemma-4B models.

Components	ALBERT		BERT		DistilBERT		RoBERTa	
	Avg.	Std. Dev	Avg.	Std. Dev	Avg.	Std. Dev	Avg.	Std. Dev
Attention Block	12.074	0.280	13.676	0.370	10.550	0.369	13.534	0.319
FFN (Intermediate)	1.510	0.441	2.765	0.192	8.764	0.442	2.565	0.321
FFN (Output)	1.848	0.050	4.944	0.343			4.875	0.401
Norm. (All)	7.317	0.531	6.216	0.845	7.135	0.369	6.352	0.623
Captured (Block)	22.748	-	27.601	-	26.449	-	27.326	-
Measured (Block)	30.627	0.562	29.802	0.607	28.551	0.448	29.797	0.642
%Capture (Block)	74.276	-	92.616	-	92.637	-	91.707	-
# of Layers	12	-	12	-	6	-	12	-
Embedding Layer	5.863	0.328	5.918	0.307	4.619	0.161	11.911	0.159
CLS + LM Head	11.687	0.455	23.666	0.979	18.396	1.009	41.081	0.907
Captured (Model)	385.068	-	387.205	-	194.320	-	410.555	-
Total	424.039	6.548	400.449	4.805	209.734	5.053	413.318	5.943
%Capture (Model)	90.810	-	96.693	-	92.651	-	99.332	-

Table 5: Comparison of FP16 Performance across ALBERT, BERT, DistilBERT, RoBERTa

Components	ALBERT - Large		BERT - Large		Distil RoBERTa		RoBERTa - Large	
	Avg.	Std. Dev	Avg.	Std. Dev	Avg.	Std. Dev	Avg.	Std. Dev
Attention Block	14.861	0.423	16.903	0.397	13.226	0.349	15.547	0.370
FFN (Intermediate)	2.928	0.743	4.877	0.456	7.413	0.396	4.249	0.115
FFN (Output)	3.939	0.710	8.093	0.462			7.741	0.443
Norm. (All)	5.814	0.652	4.171	0.872	4.274	0.711	4.721	0.671
Captured (Block)	27.542	-	34.044	-	24.913	-	32.259	-
Block	34.293	0.493	34.590	0.709	28.670	0.511	33.854	0.833
%Capture (Block)	80.314	-	98.419	-	86.896	-	95.287	-
# of Layers	24	-	24	-	6	-	24	-
Embedding Layer	6.132	0.341	7.035	0.405	11.699	0.454	12.055	0.337
CLS + LM Head	12.142	0.671	30.928	0.831	39.096	0.908	52.381	0.721
Captured (Model)	841.313	-	868.134	-	222.813	-	876.937	-
Total	919.899	16.661	855.146	17.813	235.813	3.939	870.188	4.681
%Capture (Model)	91.457	-	101.519	-	94.487	-	100.776	-

Table 6: Comparison of FP16 Performance across large variants of ALBERT, BERT, DistilBERT, RoBERTa

Components	ALBERT		BERT		DistilBERT		RoBERTa	
	Avg.	Std. Dev	Avg.	Std. Dev	Avg.	Std. Dev	Avg.	Std. Dev
Attention Block	18.573	0.517	19.155	0.436	17.772	0.412	17.870	0.327
FFN (Intermediate)	4.969	0.980	7.163	0.880	17.486	0.774	6.446	0.724
FFN (Output)	5.066	0.935	9.528	0.646			8.829	0.620
Norm. (All)	5.267	0.432	4.023	0.972	5.872	0.861	4.721	0.671
Captured (Block)	33.875	-	39.869	-	41.130	-	37.867	-
Block	40.282	0.779	42.030	0.881	46.199	0.755	40.054	1.545
%Capture (Block)	84.095	-	94.858	-	89.028	-	94.540	-
# of Layers	12	-	12	-	6	-	12	-
Embedding Layer	6.326	0.362	6.395	0.304	4.317	0.398	11.463	0.213
CLS + LM Head	20.468	0.455	48.188	0.979	39.096	0.908	74.482	1.008
Captured (Model)	510.181	-	558.945	-	320.608	-	566.588	-
Total	562.878	9.828	595.044	9.741	350.642	6.380	638.949	13.632
%Capture (Model)	90.638	-	93.933	-	91.435	-	88.675	-

Table 7: Comparison of FP32 Performance across ALBERT, BERT, DistilBERT, RoBERTa

Components	ALBERT - Large		BERT - Large		Distil RoBERTa		RoBERTa - Large	
	Avg.	Std. Dev	Avg.	Std. Dev	Avg.	Std. Dev	Avg.	Std. Dev
Attention Block	23.380	0.545	25.777	0.546	19.888	0.493	24.569	0.572
FFN (Intermediate)	8.139	1.049	9.679	0.929	7.275	0.924	9.145	0.949
FFN (Output)	8.090	0.843	12.952	0.799	9.636	0.679	12.124	0.817
Norm. (All)	4.479	0.512	4.413	0.872	3.761	0.843	4.310	0.871
Captured (Block)	44.089	-	52.821	-	40.561	-	50.148	-
Block	60.662	0.918	58.006	0.841	42.234	1.687	55.608	1.023
%Capture (Block)	72.680	-	91.061	-	96.038	-	90.183	-
# of Layers	24	-	24.000	-	6.000	-	24.000	-
Embedding Layer	5.910	0.345	7.080	0.401	14.044	0.541	13.559	0.363
CLS + LM Head	20.003	0.500	62.534	1.021	75.561	0.936	97.300	0.746
Captured (Model)	1481.807	-	1461.768	-	343.009	-	1445.440	-
Total	1562.929	9.036	1485.891	13.015	350.642	6.380	1484.479	11.134
%Capture (Model)	94.810	-	98.376	-	97.823	-	97.370	-

Table 8: Comparison of FP32 Performance across large variants of ALBERT, BERT, DistilBERT, RoBERTa

Component	32 Tokens		64 Tokens		96 Tokens		128 Tokens	
	Avg.	Std. Dev.	Avg.	Std. Dev.	Avg.	Std. Dev.	Avg.	Std. Dev.
ALBERT-Base								
Attention Block	15.094	0.404	15.930	0.354	18.582	0.319	19.861	0.468
FFN (Intermediate)	2.306	0.448	2.205	0.475	3.341	0.308	3.650	0.594
FFN (Output)	2.361	0.423	2.891	0.599	4.608	0.837	4.162	0.751
Block	35.079	0.368	37.240	0.231	44.233	0.837	45.274	0.835
Embedding Layer	6.810	0.354	6.883	0.321	7.767	0.389	6.981	0.337
CLS + LM Head	13.384	0.691	14.438	0.564	17.171	0.588	19.246	0.888
Captured (Model)	441.146	-	468.198	-	555.734	-	569.510	-
Model	482.402	9.206	505.909	4.497	571.390	2.964	598.701	5.728
% Capture (Model)	91.448	-	92.546	-	97.260	-	95.124	-
BERT- Base								
Attention Block	15.337	0.398	17.988	0.466	18.949	0.237	19.518	0.358
FFN (Intermediate)	3.472	0.393	3.695	0.186	4.628	0.494	5.626	0.653
FFN (Output)	5.210	0.374	5.947	0.361	7.450	0.290	8.774	0.272
Block	32.585	0.487	34.980	0.871	39.130	0.938	41.482	1.039
Embedding Layer	6.938	0.304	8.004	0.401	7.812	0.378	7.553	0.046
CLS + LM Head	24.876	0.906	31.174	0.798	31.497	0.615	39.772	0.897
Captured (Model)	422.840	-	458.936	-	508.874	-	545.111	-
Model	450.756	4.650	465.372	2.665	523.566	8.469	557.446	6.968
% Capture (Model)	93.807	-	98.617	-	97.194	-	97.787	-
DistilBERT								
Attention Block	12.501	0.480	13.708	0.399	14.593	0.437	14.524	0.402
FFN	10.489	0.414	10.365	0.298	13.227	0.649	14.665	0.536
Block	32.794	0.791	34.432	0.775	38.200	1.049	39.894	0.579
Embedding Layer	5.322	0.258	5.503	0.352	5.583	0.276	5.043	0.137
CLS + LM Head	20.512	0.486	27.341	0.877	26.021	0.798	34.210	0.841
Captured (Model)	222.597	-	239.438	-	260.806	-	278.618	-
Model	234.498	4.466	245.985	4.135	279.116	4.422	304.458	5.907
% Capture (Model)	94.925	-	97.338	-	93.440	-	91.513	-
RoBERTa								
Attention Block	14.996	0.398	17.547	0.438	18.979	0.426	18.479	0.332
FFN (Intermediate)	3.368	0.412	3.418	0.170	4.657	0.496	5.582	0.655
FFN (Output)	5.204	0.349	5.687	0.396	7.378	0.196	8.331	0.526
Block	32.960	0.703	35.684	1.005	40.294	1.569	41.712	1.503
Embedding Layer	12.383	0.360	14.455	0.541	15.189	0.778	12.794	0.345
CLS + LM Head	39.649	0.872	55.586	0.626	70.320	0.797	77.164	1.159
Captured (Model)	447.552	-	498.249	-	545.035	-	590.506	-
Model	470.720	5.510	506.925	5.836	553.857	4.894	594.927	7.070
% Capture (Model)	95.078	-	98.289	-	98.407	-	99.257	-

Table 9: Energy consumption trends with varying input token lengths tokens) for encoder only (Base) models

1238
1239
1240
1241

Component	32 Tokens		64 Tokens		96 Tokens		128 Tokens	
	Avg.	Std. Dev.	Avg.	Std. Dev.	Avg.	Std. Dev.	Avg.	Std. Dev.
ALBERT-Large								
Attention Block	17.730	0.472	19.696	0.645	22.541	0.529	24.279	0.435
FFN (Intermediate)	3.364	0.812	3.399	0.927	5.134	0.888	4.352	0.821
FFN (Output)	4.117	0.688	4.724	0.900	5.424	0.956	6.064	0.913
Block	39.912	0.474	44.930	1.058	51.089	0.969	55.649	0.597
Embedding Layer	7.881	0.163	7.846	0.417	7.849	0.397	7.914	0.383
CLS + LM Head	14.461	0.671	15.293	0.715	17.528	0.609	22.213	0.457
Captured (Model)	980.23	-	1101.46	-	1251.52	-	1365.71	-
Model	1029.60	5.731	1151.70	20.540	1320.18	9.018	1394.97	7.538
% Capture (Model)	95.205	-	95.638	-	94.799	-	97.903	-
BERT- Large								
Attention Block	18.641	0.275	20.986	0.405	23.092	0.670	24.792	0.351
FFN (Intermediate)	5.424	0.208	6.216	0.630	7.962	0.797	7.189	0.766
FFN (Output)	8.593	0.491	9.499	0.541	12.091	0.859	12.533	0.752
Block	37.557	0.942	41.864	1.259	48.511	1.519	50.453	1.124
Embedding Layer	7.729	0.311	8.106	0.413	8.131	0.403	8.145	0.326
CLS + LM Head	32.645	0.943	39.421	0.926	41.166	0.892	40.203	0.470
Captured (Model)	941.732	-	1052.26	-	1213.56	-	1259.22	-
Model	932.191	13.932	1060.72	28.963	1206.09	13.967	1276.73	8.691
% Capture (Model)	101.024	-	99.202	-	100.620	-	98.628	-
Distil RoBERTa								
Attention Block	15.753	0.276	17.618	0.401	18.913	0.426	18.517	0.365
FFN	3.499	0.374	3.542	0.343	4.595	0.508	5.488	0.679
Block	5.287	0.354	5.664	0.450	7.623	0.423	8.228	0.474
Embedding Layer	33.617	0.759	35.723	1.081	39.958	1.678	41.404	1.463
CLS + LM Head	12.882	0.307	14.689	0.416	14.714	0.358	12.636	0.346
Captured (Model)	39.648	0.854	55.718	0.776	70.263	0.816	77.483	1.026
Model	254.234	-	284.747	-	324.726	-	338.543	-
% Capture (Model)	270.513	4.867	292.914	3.888	319.139	5.044	341.740	5.520
RoBERTa-Large								
Attention Block	18.648	0.346	20.390	0.353	22.573	0.386	24.780	0.474
FFN (Intermediate)	5.437	0.219	5.567	0.131	7.809	0.834	7.066	0.813
FFN (Output)	8.476	0.553	9.417	0.539	11.755	0.691	12.475	0.750
Block	36.273	0.979	40.831	1.651	47.154	1.870	50.343	1.970
Embedding Layer	14.251	0.372	14.534	0.406	14.647	0.411	14.609	0.391
CLS + LM Head	51.022	0.958	69.201	1.015	89.032	1.128	94.427	0.949
Captured (Model)	935.819	-	1063.67	-	1235.36	-	1317.27	-
Model	965.216	9.228	1054.90	6.207	1238.24	10.077	1347.08	23.636
% Capture (Model)	96.954	-	100.831	-	99.768	-	97.787	-

Table 10: Energy consumption trends with varying input token lengths tokens) for encoder only (Large) models

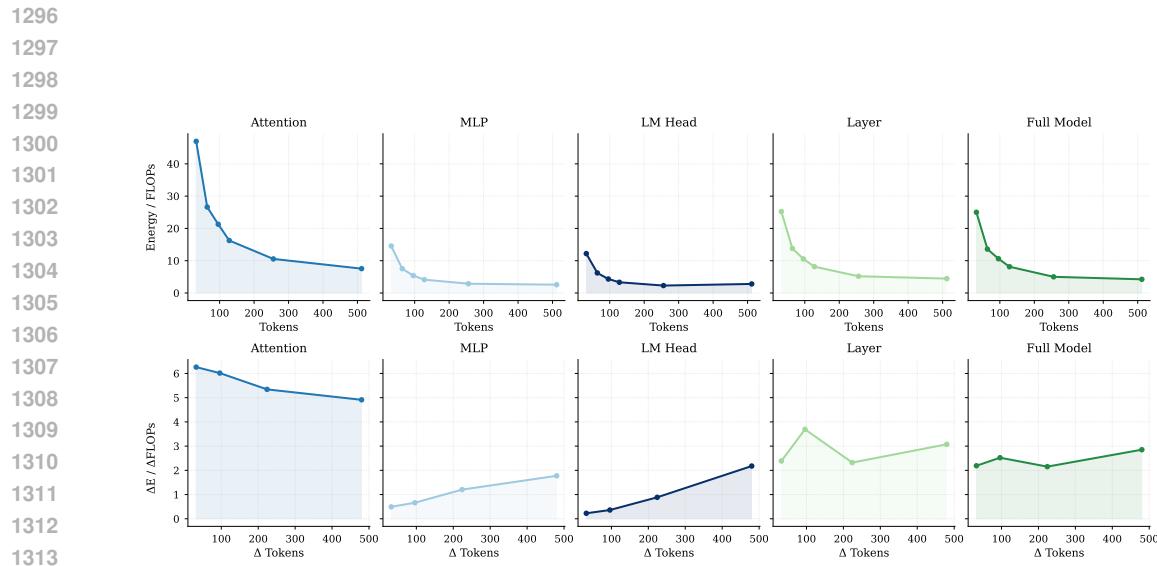


Figure 9: E/FLOP(top row) and $\Delta E/\Delta FLOP$ (bottom row) ratios for components of Gemma model across Tokens and Δ Tokens respectively. E/FLOP ratio decreases on increasing input token length but marginal energy consumed per FLOP remains nearly constant.

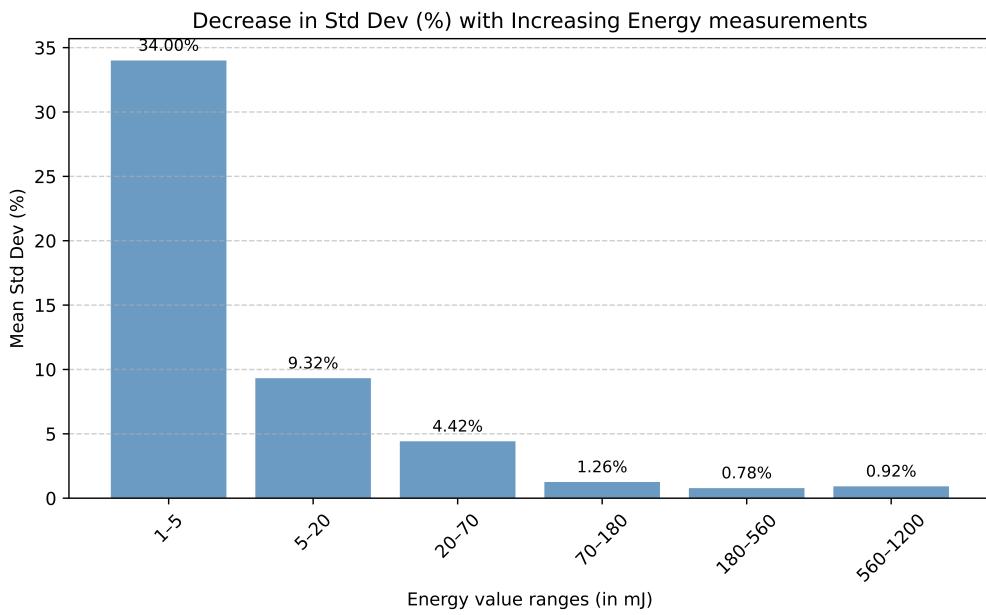


Figure 10: Depicting Decrease in Standard Deviation in our Energy Measurement Approach with increasing Energy Values.