SweetSpot: An Analytical Model for Predicting Energy Efficiency of LLM Inference

Hiari Pizzini Cavagna*
hiari.pizzinicavagna@unibo.it
University of Bologna
Bologna, BO, Italy

Andrea Proia*
andrea.proia@unibo.it
University of Bologna
Bologna, BO, Italy

Giacomo Madella
giacomo.madella@unibo.it
University of Bologna
Bologna, BO, Italy

Giovanni B. Esposito
g.esposito@unibo.it
University of Bologna
Bologna, BO, Italy

Francesco Antici
francesco.antici@unibo.it
University of Bologna
Bologna, BO, Italy

Daniele Cesarini
d.cesarini@cineca.it
Cineca
Casalecchio di Reno, BO, Italy

Zeynep Kiziltan
zeynep.kiziltan@unibo.it
University of Bologna
Bologna, BO, Italy

Andrea Bartolini
a.bartolini@unibo.it
University of Bologna
Bologna, BO, Italy

ABSTRACT

Large Language Models (LLMs) inference is central to modern AI applications, dominating worldwide datacenter workloads, making it critical to predict its energy footprint. Existing approaches estimate energy consumption as a simple linear function of input and output sequence. However, by analyzing the autoregressive structure of Transformers, which implies a fundamentally non-linear relationship between input and output sequence lengths and energy consumption, we demonstrate the existence of a generation energy minima. Peak efficiency occurs with short-to-moderate inputs and medium-length outputs, while efficiency drops sharply for long inputs or very short outputs. Consequently, we propose *SweetSpot*, an analytical model derived from the computational and memory-access complexity of the Transformer architecture, which accurately characterizes the efficiency curve as a function of input and output lengths. To assess accuracy, we measure energy consumption using TensorRT-LLM on NVIDIA H100 GPUs across a diverse set of LLMs ranging from 1B to 9B parameters, including OPT, LLaMA, Gemma, Falcon, Qwen2, and Granite. We test input and output lengths from 64 to 4096 tokens and achieve a mean MAPE of 1.79%. Our results show that aligning sequence lengths with these efficiency "sweet spots" reduce energy usage, up to 33.41×, enabling informed truncation, summarization, and adaptive generation strategies in production systems.

CCS CONCEPTS

• **General and reference** → **Measurement**; • **Hardware** → **Power and energy**; • **Computing methodologies** → **Artificial intelligence**.

*These authors contributed equally to this work.

ICPE '26, May 04–08, 2026, Florence, Italy

ACM ISBN 979-8-4007-2325-4/2026/05
https://doi.org/10.1145/3777884.3797011

KEYWORDS

LLM Energy Consumption; Energy Efficiency Modeling; LLM Sustainability

ACM Reference Format:
Hiari Pizzini Cavagna, Andrea Proia, Giacomo Madella, Giovanni B. Esposito, Francesco Antici, Daniele Cesarini, Zeynep Kiziltan, and Andrea Bartolini. 2026. SweetSpot: An Analytical Model for Predicting Energy Efficiency of LLM Inference. In *Proceedings of the 17th ACM/SPEC International Conference on Performance Engineering (ICPE '26), May 04–08, 2026, Florence, Italy*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3777884.3797011

1 INTRODUCTION

Large Language Models (LLMs) have rapidly emerged as the foundation of modern Natural Language Processing (NLP), enabling applications ranging from reasoning and summarization to dialogue systems and generative Artificial Intelligence (AI). Their adoption at scale, however, has introduced a growing concern: the substantial energy consumption associated with inference, which has been estimated to account for up to 90% of the model's life-cycle energy consumption [9]. Unlike training, which is an episodic process, inference occurs continuously in production settings, every time an LLM generates output for a user query. This shift makes inference efficiency a critical challenge for both sustainability and deployment cost.

A key contributor to inference complexity in LLMs is the attention mechanism. Since the original Transformer design [19], the standard Multi-Head Attention (MHA) has undergone several modifications aimed at improving efficiency. Variants such as Multi-Query Attention (MQA) [16] and Grouped-Query Attention (GQA) [1] reduce memory requirements by sharing Key–Value (KV) projections across heads, thereby lowering the cost of maintaining and accessing the KV cache during decoding. Recent studies [6] further highlight that attention kernels are often memory-bound rather than compute-bound, making their design a critical factor for inference efficiency at scale. These insights underscore the central role of attention in determining both the computational and energy footprint of LLM inference.

The energy consumption of LLM inference is shaped by a complex interplay of model size, architecture, and workload geometry (i.e., the distribution of input and output sequence lengths), and its characterization requires thorough empirical validation.[7] Recent studies on inference energy consumption focus only on specific models [17] or coarse-grained metrics (e.g., latency and throughput [11]), which fail to generalize and provide fine-grained inference energy profiling. Moreover, previous claims [22] on linear relation between the workload parameters (n_{in} = input tokens, n_{out} = output tokens) leads to the conclusion that every generated token incurs a constant computational and energy cost, as reflected by commercial AI providers that charge inference services based on the number of generated output tokens or a linear combination of input and output tokens. However, transformer structure and autoregressive generation process, with distinct prefill and decode phases, suggest a more complex dependency of computational cost, and consequently energy consumption, on both input and output sequence lengths. This complexity is not fully captured by current state-of-the-art analytical models.

To address these limitations, we conduct a thorough and systematic analysis of the energy consumption of a broad set of LLMs (from 1B to 9B parameters), and we explore efficiency across a wide range of input and output lengths (from 64 to 4096 tokens). Following the results of our investigation, in this paper we present the following contributions:

- The formulation of *SweetSpot*, an analytical model derived from computational and memory-access complexity of inference, capable of predicting the peak energy efficiency input and output lengths configuration. The proposed model reveals how quadratic input costs and linear decoding costs combine to produce the efficiency patterns observed in practice.
- We validate *SweetSpot* through extensive measurements across multiple LLM families, showing that it accurately captures the location of peak energy efficiency configurations ("sweet spots") as a function of input and output lengths, achieving a mean MAPE of 1.79% (± 0.61).
- We demonstrate that efficiency is not monotonically dependent on input or output sequence lengths. Instead, it exhibits distinct regimes: peak efficiency occurs for short-to-medium inputs combined with medium outputs, while efficiency sharply declines for long inputs or very short outputs, with peak energy efficiency "sweet-spots" exceeding the worst-case setting by a factor of 33.41×.

We conduct our study using TensorRT-LLM,[1] a state-of-the-art inference stack optimized for modern GPUs, running on NVIDIA H100 accelerators. Through TensorRT-LLM, we test several widely-used LLM models, such as OPT [26], LLaMA [8], Gemma [18], Falcon [2], Qwen2 [24], and Granite [12]. Doing so allows us to obtain reliable and realistic results which apply to a broad range of different models and architectures.

Our results suggest that production deployments can optimize energy efficiency by tailoring batching strategies, truncating overly long inputs, or aligning output lengths with the peak energy efficiency configurations predicted by our model.

[1] https://github.com/NVIDIA/TensorRT-LLM

The rest of the paper is organized as follows. After giving the necessary background in Section 2, we review related work in Section 3. Then, in Section 4, we present our analytical model *SweetSpot*. In Section 5 we explain our experimental methodology and illustrate the experimental results. Finally, we discuss our current limitations and future works in Section 6, and conclude in Section 7.

2 BACKGROUND

This section provides the necessary background for the analysis presented in this work. We briefly review the fundamentals of LLM architectures, their computational characteristics, and the main design principles of modern inference frameworks.

2.1 LLM Architecture

LLMs are based on the Transformer architecture [19], which has become the standard for generative AI. A transformer block is composed of two main components: the Multi-Head self-Attention mechanism (MHA) and the Feed-Forward Network (FFN), both surrounded by residual connections, layer normalization, and dropout. The attention mechanism is commonly implemented as MHA, where the hidden representation is projected into different subspaces (heads). Each head learns to capture different types of dependencies in the sequence, and their outputs are later combined. While this improves the model's expressivity and ability to represent diverse relationships, it also increases the total number of operations roughly in proportion to the number of heads. To reduce this overhead, Multi-Query Attention (MQA) was proposed in 2019 [16], modifying MHA by keeping multiple query heads but sharing a single set of Key-Value (KV) projections across all heads. This design drastically reduces memory usage and KV cache size during decoding, making inference more efficient. More recently, Grouped-Query Attention (GQA) was introduced in 2023 [1] as a generalization of MHA and MQA: keys and values are shared within groups of heads, striking a balance between the efficiency of MQA and the expressivity of MHA.

2.2 LLM Inference Computation

In large-scale models, the majority of parameters reside in the FFN layers, while the attention mechanism dominates runtime complexity due to its quadratic dependence on the input sequence length.

LLM inference is typically divided into two phases: *prefill*, where the full input sequence is processed in parallel to initialize the KV cache; and *decode*, where tokens are generated autoregressively [25]. In the prefill stage, the computational cost is dominated by the quadratic attention term, whereas in the decode stage, the cost per token is linear with respect to the sequence length, due to the reuse of cached keys and values. Despite this, the decode phase often dominates total inference time for long outputs, as each token must be produced sequentially, limiting opportunities for parallelization.

On modern GPUs, the throughput of LLM inference is not solely limited by arithmetic performance but also by memory bandwidth. Storing and retrieving the large KV cache during decoding creates significant pressure on the memory subsystem with high bandwidth requirements. For instance, in autoregressive decoding, each

generated token requires the loading of all previous keys and values from memory, resulting in attention kernels being strongly memory-bound, particularly for long contexts.

2.3 Inference Frameworks

When executing LLM inference on production hardware, several software for inference run-time exist, such as TensorRT-LLM [14], vLLM [10] and DeepSpeed [3], each offering different levels of optimization and targeting different deployment constraints. They vary substantially in how they manage memory, parallelize attention, and schedule kernels. TensorRT-LLM focuses on graph-level optimizations and low-level kernel fusion, while vLLM is an open-source inference system explicitly tailored for high-throughput LLM serving. Its main contribution, the PagedAttention mechanism, improves memory efficiency by partitioning KV caches, thereby enabling scalable batch inference with reduced memory overhead. DeepSpeed, developed by Microsoft, provides a distributed training and inference framework aimed at maximizing scalability using techniques such as the Zero Redundancy Optimizer [15].

2.3.1 ***TensorRT-LLM.*** In particular, TensorRT-LLM is NVIDIA's specialized inference framework designed for optimizing LLM serving on modern GPUs. It builds upon the TensorRT optimization stack, extending it with kernels and execution strategies tailored for Transformer-based architectures. The framework provides a toolchain that converts pretrained models into highly optimized TensorRT engines. This process includes parsing the original model graph, applying operator fusion, and lowering precision to FP16, BF16, or INT8 when supported. Once converted, the engine is serialized and can be executed through the TensorRT runtime, which is tightly integrated with CUDA and NVIDIA's kernel libraries. Compared to general-purpose libraries, TensorRT-LLM provides a lower-level execution model where computational and memory operations are highly optimized for modern GPUs (e.g., NVIDIA's Hopper architecture).

Beyond these system-level design advantages, recent empirical evidence further motivates the use of TensorRT-LLM. In particular, the authors of [13] present a comparative analysis of the frameworks mentioned above, and find that vLLM and TensorRT-LLM consistently achieve the most energy-efficient token generation among the compared frameworks, especially under high-concurrency and high-throughput workloads. Component-level measurements further show that both vLLM and TensorRT-LLM consume only about 5% of the GPU energy compared to a general-purpose serving framework such as Transformers [23]. This result highlights how their hardware-aware optimizations translate directly into power savings. Collectively, these results show that TensorRT-LLM is not only engineered for performance but is also empirically validated as one of the most energy-efficient inference engines available. Based on these considerations, as well as to obtain a production-grade figure of merit, we choose TensorRT-LLM as the inference run-time in this paper. Future work will extend the analysis to different frameworks.

2.3.2 ***In-Flight Batching.*** A key run-time optimization employed by TensorRT-LLM to maximize throughput under dynamic workloads is its *in-flight batching* strategy. Unlike static batching, where requests must be grouped prior to execution and processed synchronously, in-flight batching enables the run-time to continuously merge and schedule requests that are already undergoing inference. This mechanism is particularly well-suited for autoregressive LLM decoding, where sequences progress token by token and naturally diverge in length. At a high level, TensorRT-LLM decouples request admission from kernel execution. Incoming inference requests are first tokenized and then admitted into an active set of sequences maintained by the run-time scheduler. During each decoding iteration, TensorRT-LLM dynamically forms micro-batches composed of all active sequences that are ready to generate the next token. Newly arriving requests can be inserted into this active set without waiting for the completion of previous batches, while completed sequences are removed as soon as they reach their termination condition. As a result, GPU execution remains continuously occupied, reducing idle cycles that would otherwise arise from batch fragmentation. From a system perspective, in-flight batching trades strict per-request latency determinism for significantly improved throughput and hardware utilization, being especially effective in production serving scenarios, where request arrival times and sequence lengths are inherently unpredictable.

The behavior of in-flight batching is managed by user-configurable limits, namely `max_batch_size` and `max_num_tokens`. The `max_batch_size` parameter specifies the maximum number of active requests that can be scheduled concurrently in a decoding iteration, effectively bounding the degree of request-level parallelism. In contrast, `max_num_tokens` constrains the total number of tokens processed in a single micro-batch, aggregating across all active sequences. This token-level limit is particularly important for controlling GPU memory usage per iteration.

3 RELATED WORK

Several recent works have characterized and modeled the energy footprint of LLM inference. For instance, [7] presents a systematic analysis of inference energy-efficiency optimizations across diverse NLP and generative AI workloads, considering input–output token distributions, batching strategies, GPU hardware, software frameworks, and decoding methods. Their work shows that real-world energy consumption is highly sensitive to workload geometry and software–hardware configurations. Authors of [13] characterize the energy consumption of different inference engines (Transformers, vLLM, DeepSpeed, and TensorRT-LLM). They compute the Energy-per-Token metric E_{tok} as:

$$E_{\text{tok}} = \frac{E_{\text{tot}}}{n_{\text{out}}} \tag{1}$$

where E_{tot} is the total energy consumption during inference and n_{out} the total generated output tokens. In [21], they investigate the trade-offs between accuracy and energy consumption when applying test-time compute strategies. They propose an energy-aware routing mechanisms to guide sustainable model selection and inference, defining E_{tok} to account for the number of input tokens n_{in}, by expressing it as:

$$E_{\text{tok}} = \frac{E_{\text{tot}}}{n_{\text{out}} + n_{\text{in}}} \tag{2}$$

Table 1: Analytical Model Set

Model	Formula
Baseline 1 [13]	$E_{\text{tok}} = \theta_0$
Baseline 2 [13]	$E_{\text{tok}}(n_{\text{out}}) = \theta_0 + \frac{\theta_1}{n_{\text{out}}}$
Baseline 3 [21]	$E_{\text{tok}}(n_{\text{in}}, n_{\text{out}}) = \theta_0 + \frac{\theta_1}{n_{\text{in}}+n_{\text{out}}}$
Baseline 4 [22]	$E_{\text{tok}}(n_{\text{in}}, n_{\text{out}}) = \theta_0 + \frac{\theta_1 n_{\text{in}}}{n_{\text{out}}} + \theta_2 n_{\text{in}}$
SweetSpot (FLOPs-only) (ours)	$E_{\text{tok}}(n_{\text{in}}, n_{\text{out}}) = \theta_0 + \frac{\theta_1 n_{\text{in}}^2}{n_{\text{out}}} + \theta_2 n_{\text{in}} + \frac{\theta_3 n_{\text{in}}}{n_{\text{out}}} + \theta_4 n_{\text{out}}$
SweetSpot (ours)	$E_{\text{tok}}(n_{\text{in}}, n_{\text{out}}) = \theta_0 + \frac{\theta_1 n_{\text{in}}^2}{n_{\text{out}}} + \theta_2 n_{\text{in}} + \frac{\theta_3 n_{\text{in}}}{n_{\text{out}}} + \theta_4 n_{\text{out}} + \frac{\theta_5}{n_{\text{out}}}$

Finally, in [22] they characterize the energy consumption and run-time behavior of a set of LLMs using A100 GPUs, proposing workload-based energy models that, a given model predicts consumption as a function of only the input and output tokens. To the best of our knowledge, this is the only work that proposes a model to describe the total energy consumption as a function of input and output token as follows:

$$E_{\text{tot}} = \theta_0 n_{\text{in}} + \theta_1 n_{\text{out}} + \theta_3 n_{\text{in}} n_{\text{out}} \tag{3}$$

In contrast to prior studies, we introduce *SweetSpot*, an analytical model of LLM inference energy consumption derived from both computational and memory-access complexity. This model enables us to characterize the non-linear interplay between input and output sequence lengths. To assess it, we conduct a systematic study of inference energy usage using TensorRT-LLM on NVIDIA H100 GPUs, across a diverse set of models and a wide range of sequence lengths. Our formulation makes explicit how quadratic input-processing costs and linear decoding costs combine to shape the efficiency patterns observed in practice, and we further extend the model to incorporate memory-access effects. Through extensive evaluation across multiple model families, we show that *SweetSpot* closely matches measured energy trends and correctly identifies where efficiency peaks occur as a function of input and output lengths. Finally, we show that efficiency does not vary monotonically with sequence length; instead, it follows distinct regimes, with clear "sweet-spots" configurations and degradation zones depending on the balance between input size and output length.

4 SWEETSPOT: MODEL FORMULATION AND ENERGY EFFICIENCY PREDICTION

In this section, we present *SweetSpot*, our analytical model for predicting LLM inference energy efficiency, explicitly analyzing the distinct computational characteristics of the prefill and decode phases. We first introduce a version of *SweetSpot* based solely on the computational complexity of inference, and subsequently extend it to incorporate memory-access complexity. To assess the validity of our approach, we compare *SweetSpot* against four baselines, summarized in Table 1. From [13] we derive Baseline 1, which simply describes the E_{tok} as constant. Moreover, we formalize a parametrized model that relies solely on the number of generated tokens, n_{out}, to characterize E_{tok}, referred to as Baseline 2. The third baseline, Baseline 3, expands on this by also including the number of input tokens, n_{in}, as introduced in Equation 2. This formulation allows us to establish a relationship between E_{tok} and the total sequence length. Finally, Baseline 4 is derived from Equation 3, normalizing the equation by n_{out}.

4.1 SweetSpot Model (FLOPs-only)

To evaluate the energy efficiency of our LLMs, we derived an initial analytical model grounded exclusively in the computational complexity of inference. The model accounts for both the *prefill* phase, where the entire input sequence is processed in parallel, and the *decode* phase, where tokens are generated autoregressively using the cached KV states. In the following, we present the derivation of the formulas for each phase and then combine them into a unified expression. During the prefill stage, the model processes an input sequence of length n_{in} in parallel. For a single Transformer layer with hidden size d, the computational cost can be decomposed into two main contributions: the Self-Attention Mechanism and the Feed-Forward Network (FFN). We neglect the cost of scaling, softmax, layer normalization, dropout, and bias additions, which are negligible w.r.t. the cost of attention and FFN.

Moreover, we adopt the standard Floating-point Operations (FLOPs), denoted by F, accounting convention where one multiply–accumulate operation is counted as two FLOPs (one multiplication and one addition).

4.1.1 *Prefill Stage FLOPs.*

Attention. The attention mechanism involves the following matrix multiplications:

Input projections to Query, Key, Value (Q, K, V): the input $X \in \mathbb{R}^{n_{\text{in}} \times d}$ is multiplied by three weight matrices $W_Q, W_K, W_V \in \mathbb{R}^{d \times d}$:

$$F_{\text{proj}} = 3 \cdot (2 n_{\text{in}} d^2) = 6 n_{\text{in}} d^2$$

This corresponds to three dense matrix multiplications of size $n_{\text{in}} \times d$ by $d \times d$.

Attention score computation $QK^\top$: multiplication of $Q \in \mathbb{R}^{n_{\text{in}} \times d}$ with $K^\top \in \mathbb{R}^{d \times n_{\text{in}}}$:

$$F_{QK^T} = 2 n_{\text{in}}^2 d$$

Weighted sum with V: multiplication of the $n_{\text{in}} \times n_{\text{in}}$ attention matrix with $V \in \mathbb{R}^{n_{\text{in}} \times d}$:

$$F_V = 2 n_{\text{in}}^2 d$$

Final output projection: multiplication with $W_O \in \mathbb{R}^{d \times d}$:

$$F_{\text{out}} = 2 n_{\text{in}} d^2$$

Summing these terms yields the attention cost per layer:

$$F_{\text{att}} = 8 n_{\text{in}} d^2 + 4 n_{\text{in}}^2 d$$

FFN. The feed-forward network expands the hidden dimension by a factor of 4, with two dense matrix multiplications per token:

$$F_{\text{FFN}} = 8 n_{\text{in}} d^2 + 8 n_{\text{in}} d^2 = 16 n_{\text{in}} d^2$$

corresponding to the attention output multiplication with W_1 and the subsequent W_2 projection back to dimension d.

Total Prefill. For one layer:

$$F_{\text{prefil, layer}} = F_{\text{att}} + F_{\text{FFN}} = 24 n_{\text{in}} d^2 + 4 n_{\text{in}}^2 d$$

For an L-layer model:

$$F_{\text{prefill}} = L \cdot \left(24 n_{\text{in}} d^2 + 4 n_{\text{in}}^2 d\right)$$

4.1.2 ***Decode Stage FLOPs.*** At decoding step t, the model has access to $n_{\text{in}} + t - 1$ cached tokens. For each new token, the per-layer costs are:

Attention. Projections for Q, K, V, and output: multiplications with $d \times d$ weight matrices:

$$F_{\text{proj+out}} = 8d^2$$

Attention with cached keys and values: multiplication of $Q \in \mathbb{R}^{1\times d}$ with cached $K^\top \in \mathbb{R}^{d\times(n_{\text{in}}+t-1)}$ and multiplication of resulting attention scores with $V \in \mathbb{R}^{(n_{\text{in}}+t-1)\times d}$:

$$F_{QK^T+V}(t) = 4(n_{\text{in}} + t - 1)d$$

FFN. Per token, the FFN again involves two $d \times d$ multiplications:

$$F_{\text{FFN}} = 16d^2$$

Total per Token. The per-token cost for one layer is therefore:

$$\begin{aligned} F_{\text{decode, layer}}(t) &= F_{\text{proj+out}} + F_{QK^T+V} + F_{\text{FFN}} \\ &= 24d^2 + 4(n_{\text{in}} + t - 1)d \end{aligned}$$

Total Decode. Summing over n_{out} generated tokens and multiplying by L layers:

$$\begin{aligned} F_{\text{decode}} &= L \sum_{t=1}^{n_{\text{out}}} \left(24d^2 + 4(n_{\text{in}} + t - 1)d\right) \\ &= L\big(24 n_{\text{out}} d^2 + 4d(n_{\text{out}} n_{\text{in}} + \tfrac{n_{\text{out}}(n_{\text{out}}-1)}{2})\big) \end{aligned}$$

4.1.3 ***Total Inference FLOPs.*** By summing the Prefill and Decode FLOPs, we obtain:

$$F_{\text{total}} = F_{\text{prefill}} + F_{\text{decode}} = L\left[24d^2(n_{\text{in}} + n_{\text{out}}) + 4d\big(n_{\text{in}}^2 + n_{\text{out}} n_{\text{in}} + \tfrac{n_{\text{out}}(n_{\text{out}}-1)}{2}\big)\right]$$

We factor out the common multiplier $2Ld$ and manipulate the formula to spot individual contributors. This compact form is convenient for later normalization to build the final model.

$$F_{\text{total}} = 2\,L\,d\,\left(12\,d\,n_{\text{in}} + 12\,d\,n_{\text{out}} + 2\,n_{\text{in}}^2 + 2\,n_{\text{out}}\,n_{\text{in}} + n_{\text{out}}^2 - n_{\text{out}}\right)$$

We note that the external factors L and d depends only on model size (depth and width) and are constant for a given model. Since our goal is to capture how efficiency varies as a function of input length n_{in} and output length n_{out}, we remove these constant multipliers and express the energy as a linear combination of polynomial terms in terms of n_{in} and n_{out}. Aggregating similar terms, we obtain:

$$E_{\text{tot}}(n_{\text{in}}, n_{\text{out}}) = \theta_0\, n_{\text{out}} + \theta_1\, n_{\text{in}}^2 + \theta_2\, n_{\text{out}}\, n_{\text{in}} + \theta_3\, n_{\text{in}} + \theta_4\, n_{\text{out}}^2$$

This formulation highlights the individual contributions of input length, output length, and their interaction. The learned parameters θ_i capture the proportionality between the theoretical FLOP terms and the actual measured energy in Joules.

To analyze the computational cost on a *per-output-token* basis, we divide the total energy by the number of generated tokens n_{out}. This provides the average energy required to produce a single token, which is useful for comparing efficiency across different input–output configurations.

$$E_{\text{tok}}(n_{\text{in}}, n_{\text{out}}) = \theta_0 + \frac{\theta_1\, n_{\text{in}}^2}{n_{\text{out}}} + \theta_2\, n_{\text{in}} + \frac{\theta_3\, n_{\text{in}}}{n_{\text{out}}} + \theta_4\, n_{\text{out}} \tag{4}$$

This normalized form highlights how the cost of processing the input sequence n_{in} is amortized across the output tokens, while the terms proportional to n_{out} reflect the per-token cost of decoding.

4.2 SweetSpot Model

While FLOPs capture the arithmetic workload of inference, modern accelerators increasingly operate in a regime where energy consumption is dominated not by computation but by memory traffic, in particular, the movement of activations, weights, and KV-cache tensors across the memory hierarchy [25]. To accurately model energy, we therefore incorporate the memory-access cost alongside the computational cost derived in the previous section. In the following subsections, we present an analytical decomposition of the memory contribution during both the prefill and decode phases, denoted by M. As before, we analyze a single Transformer layer with hidden dimension d and input length n_{in}, and n_q number of heads. We again omit the negligible contributions of bias reads, layer-normalization reads/writes, and softmax.

4.2.1 ***Prefill Stage Memory Operations.*** During prefill, the model processes the entire sequence in parallel, producing input projections and attention tensors. Memory traffic arises from reading inputs, writing intermediate projected representations and attention matrices. For one layer, the total memory movement is decomposed as follows.

Attention. The attention mechanism involves the following memory transfers:
Input projections to Q, K, V: reading $X \in \mathbb{R}^{n_{\text{in}}\times d}$ and writing projected tensors:

$$M_{\text{proj}} = 2n_{\text{in}}d + d^2$$

Attention score computation $QK^\top$: reading $Q \in \mathbb{R}^{n_{\text{in}}\times d}$ and $K^\top \in \mathbb{R}^{d\times n_{\text{in}}}$ and writing the $n_{\text{in}} \times n_{\text{in}}$ attention matrix:

$$M_{QK^T} = 2n_{\text{in}}d + n_{\text{in}}^2 n$$

Weighted sum with V: reading the attention matrix and $V \in \mathbb{R}^{n_{\text{in}}\times d}$ (n_q = number of heads):

$$M_V = 2n_{\text{in}}d + n_{\text{in}}^2 n_q$$

Final output projection: reading the attention output ($n_{\text{in}} \times d$) and writing the output representation:

$$M_{\text{out}} = 2n_{\text{in}}d + d^2$$

Summing these terms gives the total attention memory per layer during prefill:

$$M_{\text{att}} = 8n_{\text{in}}d + 2d^2 + 2n_{\text{in}}^2 n_q$$

FFN. The feed-forward network reads the input activations, accesses its two weight matrices, and writes the intermediate expanded representation:

$$M_{\text{FFN}} = 2n_{\text{in}}d + 8d^2$$

Total Prefill. For one layer:

$$M_{\text{prefil, layer}} = M_{\text{att}} + M_{\text{FFN}} = 10n_{\text{in}}d + 10d^2 + 2n_{\text{in}}^2 n_q$$

For an L-layer model:

$$M_{\text{prefill}} = L\left(10n_{\text{in}}d + 10d^2 + 2n_{\text{in}}^2 n_q\right)$$

4.2.2 ***Decode Stage Memory Operations.*** Similarly to the FLOPs Decoding stage, at each step t, we describe the per-layer memory operations as follows.

Attention. Projections for Q, K, V, and output: multiplications with $d \times d$ weight matrices:

$$M_{\text{proj+out}} = 2(2d + d^2)$$

Attention with cached keys and values: multiplication of $Q \in \mathbb{R}^{1\times d}$ with cached $K^\top \in \mathbb{R}^{d\times(n_{\text{in}}+t-1)}$ and multiplication of resulting attention scores with $V \in \mathbb{R}^{(n_{\text{in}}+t-1)\times d}$:

$$M_{QK^T+V}(t) = 2((n_q + d)(n_{\text{in}} + t - 1) + d)$$

FFN. Per token, the FFN again involves two $d\times d$ multiplications:

$$M_{\text{FFN}} = 2d + 8d^2$$

Total per Token. The per-token cost for one layer is therefore:

$$\begin{aligned} M_{\text{decode, layer}}(t) &= M_{\text{proj+out}} + M_{QK^T+V} + M_{\text{FFN}} \\ &= 7d + 10d^2 + (n_q + d)(n_{\text{in}} + t - 1) \end{aligned}$$

Total Decode. Summing over n_{out} generated tokens and multiplying by L layers:

$$\begin{aligned} M_{\text{decode}} &= L\sum_{t=1}^{n_{\text{out}}} \left(7d + 10d^2 + (n_q + d)(n_{\text{in}} + t - 1)\right) \\ &= L\big(7dn_{\text{out}} + 10d^2 n_{\text{out}} + (n_q + d)(n_{\text{in}}n_{\text{out}} + \tfrac{n_{\text{out}}(n_{\text{out}}-1)}{2})\big) \end{aligned}$$

4.2.3 ***Total Memory Operations.*** By summing the Prefill and Decode Memory Operations, we obtain:

$$\begin{aligned} M_{\text{total}} &= M_{\text{prefill}} + M_{\text{decode}} \\ &= L\Big[10n_{\text{in}}d + 10d^2 + 2n_{\text{in}}^2 n_q \\ &\quad + n_{\text{out}}\big(7d + 10d^2 + (n_q + d)(n_{\text{in}} + \tfrac{(n_{\text{out}}-1)}{2})\big)\Big] \end{aligned}$$

As for the FLOPs-only model, we express the energy consumption as a linear combination of polynomial terms in n_{in} and n_{out}:

$$E_{\text{tot}}(n_{\text{in}}, n_{\text{out}}) = \theta_0\, n_{\text{out}} + \theta_1\, n_{\text{in}}^2 + \theta_2\, n_{\text{out}}\, n_{\text{in}} + \theta_3\, n_{\text{in}} + \theta_4\, n_{\text{out}}^2 + \theta_5$$

Relative to the E_{tot} of the FLOPs model, this formulation introduces an additional constant term, θ_6. This term accounts for the contribution arising from the $10d^2$ component in the M_{prefill} expression, which is independent of both n_{in} and n_{out}. Analyzing the energy consumption *per-output-token*, we obtain:

$$E_{\text{tok}}(n_{\text{in}}, n_{\text{out}}) = \theta_0 + \frac{\theta_1\, n_{\text{in}}^2}{n_{\text{out}}} + \theta_2\, n_{\text{in}} + \frac{\theta_3\, n_{\text{in}}}{n_{\text{out}}} + \theta_4\, n_{\text{out}} + \frac{\theta_5}{n_{\text{out}}} \quad (5)$$

4.3 Peak Energy Efficiency Prediction

Using the formulation of *SweetSpot* introduced in Equation 5, we identify peak energy-efficiency configurations by considering the per-token energy as a function of the number of generated tokens n_{out}. To this end, we compute the derivative of E_{tok} with respect to n_{out} and set it equal to zero:

$$\frac{\partial E_{\text{tok}}}{\partial n_{\text{out}}} = -\frac{\theta_1 n_{\text{in}}^2 + \theta_3 n_{\text{in}} + \theta_5}{n_{\text{out}}^2} + \theta_4 = 0$$

Solving for n_{out}, we obtain:

$$\frac{\theta_1 n_{\text{in}}^2 + \theta_3 n_{\text{in}} + \theta_5}{n_{\text{out}}^2} = \theta_4 \quad \Rightarrow \quad n_{\text{out}}^2 = \frac{\theta_1 n_{\text{in}}^2 + \theta_3 n_{\text{in}} + \theta_5}{\theta_4} \quad \Rightarrow$$

$$\boxed{n_{\text{out}}^* = \sqrt{\frac{\theta_1 n_{\text{in}}^2 + \theta_3 n_{\text{in}} + \theta_5}{\theta_4}}} \quad (6)$$

This result shows that the per-token energy is minimized when the number of generated tokens is equal to the square root of a combination of the input sequence length and its quadratic contribution, normalized by the linear per-token decoding cost θ_4.

5 EXPERIMENTAL STUDY

This section details the experimental study conducted to validate *SweetSpot* and to evaluate the energy consumption of LLMs inference on modern hardware. We describe the experimental design, including the monitoring framework, LLM selection, metrics, datasets, engine configurations, benchmarking procedure, and system setup. We then report and analyze the experimental results, evaluating the energy efficiency of the LLM test set and the accuracy of our proposed analytical model *SweetSpot*.

5.1 Experimental Design

This subsection formalizes the experimental protocol, describing the measurement pipeline and design choices adopted to ensure controlled and reproducible evaluation across all experiments.

5.1.1 ***Monitoring Framework.*** To collect and analyze GPU metrics during LLM inference, we employ an updated version of ExaMon [4], based on IoTdb Database [20]. In particular, hardware metrics such as `nvmlDeviceGetPowerUsage` and `NVML_CLOCK_GRAPHICS` are sampled using the NVIDIA Management Library Python API (pynvml [5]). These metrics are sampled at *500 ms* intervals, published through the ExaMon plugin, and stored in the ExaMon database.

5.1.2 ***LLM Selection.*** We consider a diverse set of LLMs, ranging from 1B to 9B parameters, as summarized in Table 2. Beyond model size, the selection includes architectures with different design choices and attention mechanisms, including Multi-Head Attention (MHA) and Grouped-Query Attention (GQA). For GQA, we also account for different KV grouping ratios e.g. GQA (4×) denotes four query heads sharing a single KV head. To facilitate comparison, the models are grouped into three size-based families, ranging from XS models (1B–1.5B parameters) to M models (6.7B–9B parameters).

Table 2: LLM test set grouped into size-based families (1B–9B parameters), with architectural details for each model.

Name	Size (B)	Heads-Attention	Hidd. Size	Layers
XS models (From 1B to 1.5B)				
Llama 3.2	1	32 - GQA (4x)	2048	16
OPT	1.3	32 - MHA	2048	24
Qwen 2	1.5	12 - GQA (6x)	1536	28
S models (From 2B to 3B)				
Gemma 2	2	8 - GQA (2x)	2304	26
OPT	2.7	32 - MHA	2560	32
Llama 3.2	3	24 - GQA (4x)	3072	28
Granite	3	32 - MHA	2048	32
M models (From 6.7B to 9B)				
OPT	6.7	32 - MHA	4096	32
Qwen 2	7	28 - GQA (7x)	3584	28
Falcon-RW	7	64 - MHA	4096	36
Granite	8	32 - GQA (4x)	4096	36
Llama 3.1	8	32 - GQA (4x)	4096	32
Gemma 2	9	16 - GQA (2x)	3584	42

5.1.3 ***Metrics and Measurement.*** We quantify the energy consumption E_{tot} as the discrete integration of the mean instantaneous GPU power, measured as described in Section 5.1.1 over the measurement interval Δt, according to the following expression:

$$E_{\text{tot}} = \sum_i \left(\frac{P_i + P_{i+1}}{2} \right) \Delta t$$

where P_i denotes the power usage recorded at each time step i. We empirically obtain the Energy-per-Token (E_{tok}) by normalizing the energy consumption by the total amount of output tokens n_{out}:

$$E_{\text{tok}} = \frac{E_{\text{tot}}}{n_{\text{out}}} \quad \left[\frac{\text{Joules}}{\text{Token}} \right]$$

For improved interpretability, we additionally define the LLM energy efficiency as the number of output tokens generated per unit of energy. Formally, this metric corresponds to the reciprocal of E_{tok}:

$$E_{\text{eff}} = \frac{n_{\text{out}}}{E_{\text{tot}}} \quad \left[\frac{\text{Tokens}}{\text{Joule}} \right]$$

This measure captures the productivity of the model with respect to its energy budget, facilitating comparisons between models.

5.1.4 ***Dataset Generation.*** To benchmark TensorRT-LLM under controlled and reproducible conditions, we generated synthetic request datasets using the official utility script provided with TensorRT-LLM. For each dataset, token sequences were sampled using the Llama-3.1-8B-Instruct tokenizer, with input and output lengths drawn from fixed distributions defined by a specified mean and zero variance. This allowed us to precisely control the number of input and output tokens per request. We generated the dataset.[2] for all combinations of input and output lengths ranging from 64 to 4096 tokens (in powers of two), and for different workload sizes with the number of requests set to 10, 100, and 1000.

[2]The dataset is publicly available on Zenodo: doi.org/10.5281/zenodo.18714476

5.1.5 ***Engines Configuration.*** All TensorRT-LLM engines used in our benchmarks were built using the `trtllm-bench build`, which converts model checkpoints into TensorRT-LLM engines. Engines were constructed with tensor parallelism and pipeline parallelism both set to one, ensuring single-GPU execution and isolating inference performance from inter-device communication effects. The maximum supported sequence length was set to 8192 tokens, allowing all benchmarked input–output combinations to execute within a single engine configuration without triggering engine rebuilds. Additionally, we configured the `max_batch_size` to 1024 and the `max_num_tokens` to 8192. This approach ensures that performance differences observed across experiments are attributable to workload characteristics rather than changes in engine structure or parallelization strategy.

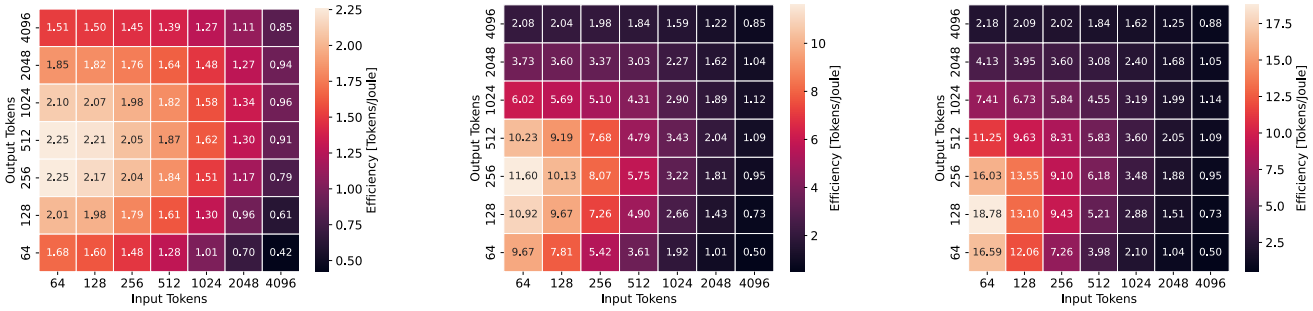
5.1.6 ***Benchmark Execution.*** Benchmark runs were executed with `trtllm-bench` in *throughput* mode, a dedicated TensorRT-LLM utility for evaluating LLM performance on a given dataset, designed to maximize sustained token generation rate under a fixed workload. Each run pairs a prebuilt engine with a corresponding synthetic dataset, ensuring that model configuration and workload characteristics remain constant throughout the measurement. We first assessed different number of requests (10, 100, and 1000) using Falcon-RW 7B. Subsequently, we evaluated the LLMs listed in Table 2 by fixing the number of requests to 1000, while varying the input and output sequence lengths from 64 to 4096 tokens. By reusing the same engine across datasets and varying only the dataset parameters, the reported results isolate the impact of input/output length and request count on inference throughput.

5.1.7 ***System Setup.*** All experiments were conducted on a single-node server equipped with an *Intel® Xeon® Platinum 8480+* processor, belonging to the Sapphire Rapids family. In addition, the system hosts four NVIDIA H100 Tensor Core GPUs, connected through NVLink. The server is equipped with 32x64 GB of DDR5 RAM operating at a frequency of 4800 MHz. The system runs on a *Dell 00G41X Version A02* motherboard. The operating system is Alma Linux 9.5 with Linux kernel 5.14. The cooling infrastructure relies on a rack-mounted air cooling system with redundant fans, which operated at stable speeds of 5160 RPM during the experiments.

5.2 Experimental Results

In this subsection we discuss the results of our experimental study, evaluating the energy efficiency of the tested LLMs and the accuracy of our analytical model *SweetSpot*, using it to predict the peak energy efficiency spots.

5.2.1 ***Number of Requests Selection.*** To assess the impact of the request count N_{req} (i.e. the number of queries used for evaluation) on energy efficiency, we considered multiple workload sizes. We excluded the single-request setting because our monitoring framework introduces a non-negligible timing error, preventing reliable measurements in that regime where execution times are shorter. Figure 1 shows the energy efficiency for the Falcon-RW 7B model (the least efficient LLM of our set) across different workload size (10,100 and 1000). Each plot reports the average energy efficiency for different LLM inference run, each with a different n_{out} (x-axis) and n_{in} (y-axis) configurations. As shown in the figure, the



(a) $N_{\text{req}} = 10$. **(b)** $N_{\text{req}} = 100$. **(c)** $N_{\text{req}} = 1000$.

Figure 1: Energy efficiency $\frac{\text{Tokens}}{\text{Joule}}$ of Falcon-RW 7B with different numbers of requests (10, 100, 1000).

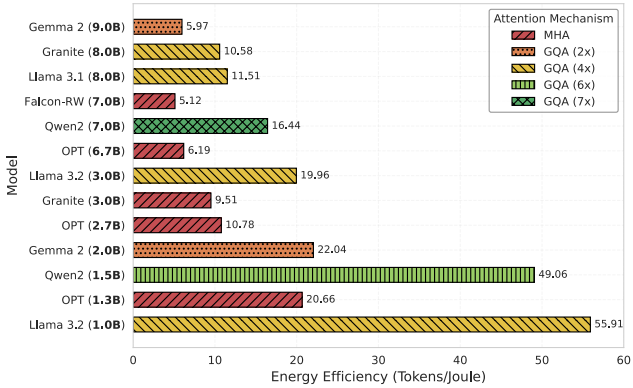


Figure 2: LLMs average Energy Efficiency across the entire dataset.

workload size strongly affects both the absolute efficiency values and their overall trends. In particular, the efficiency peak shifts toward larger output lengths when using fewer requests, a behavior driven by the reduced number of tokens processed per inference step. Furthermore, the spread between the highest and the lowest efficiency values narrows for smaller workloads. Specifically, with 10 requests, E_{eff} ranges from 0.42 to 2.25, which consists in an increase of about 6×. In contrast, with 1000 requests, the values range from 0.50 to 18.78, corresponding to a significantly wider gap of approximately 37.6×.

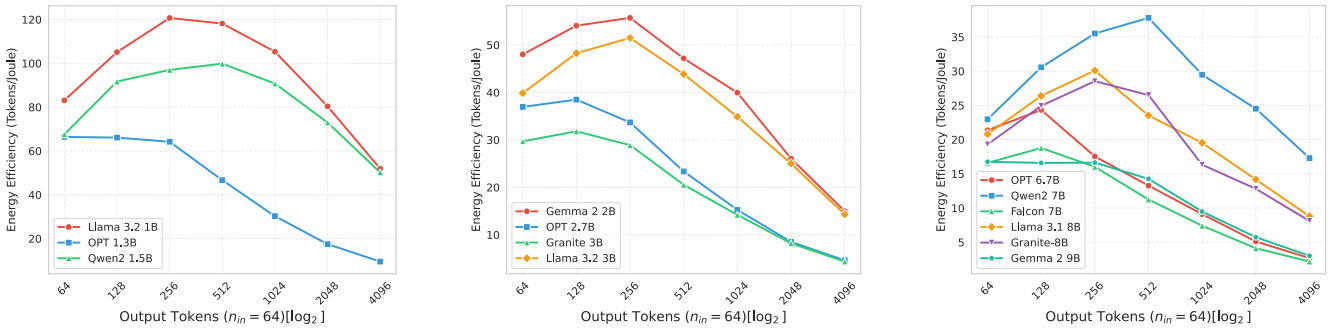
Finally, smaller workloads amplify the relative impact of measurement overhead and necessitate shorter sampling intervals, which can further amplify measurement noise and result in less reliable estimates. For these reasons, in the subsequent experiments we focus on the saturation scenario and fix N_{req} to 1000. A more detailed analysis of how workload size interacts with batch size is provided in Section 6.2.

5.2.2 ***Overall LLMs Energy Efficiency.*** In Figure 2, we report the average energy efficiency measured across the entire dataset for each LLM in our test (see Table 2), highlighting the attention mechanism adopted by each model. Among the evaluated models, Llama 3.2 1B achieves the highest overall energy efficiency, with an average score of 55.91. In contrast, Falcon-RW 7B exhibits the lowest efficiency, with an average value of 5.12.

Model size emerges as a primary factor influencing energy efficiency. In general, larger models tend to be less energy efficient than smaller ones. However, parameter count alone does not fully explain the observed differences. The attention mechanism also plays a significant role. In particular, models employing MHA are consistently less energy efficient than comparably sized models adopting GQA. Unlike MHA, where each attention head maintains independent Key–Value (KV) projections, GQA allows multiple query heads to share the same KV projections, which reduces the size of the KV cache and the number of memory accesses required at each decoding step, leading to lower bandwidth pressure and improved energy efficiency. For example, OPT 1.3B (employing MHA) is approximately 42% less efficient than Qwen2 1.5B (GQA 6x), achieving average efficiencies of 20.66 and 49.06, respectively, despite having fewer parameters. A similar pattern is observed when comparing OPT 6.7B and Qwen2 7B. Moreover, higher GQA grouping ratios appear to correlate with improved efficiency. For instance, Qwen 7B (GQA 7x) is only about 18% less efficient than Llama 3.2 3B (GQA 4x), despite having more than twice the number of parameters. This suggests that architectural choices in the attention mechanism can partially mitigate the energy cost typically associated with increased model size.

While the attention mechanism is a critical determinant of energy efficiency, it is not the only architectural factor involved. The number of layers, hidden dimensionality, intermediate projection sizes, and other architectural design choices also substantially impact overall efficiency. A systematic analysis of how these architectural components interact with energy consumption is left for future work.

5.2.3 ***Observation of Peak Energy Efficiency Spots.*** In Figure 3, we show the energy efficiency of each LLM in our test set, reported in Table 2, considering a fixed input length of $n_{\text{in}} = 64$, the input length which shows the maximum efficiency, and varying the output length. The sub-figures correspond to different LLM size families (XS, S, M), and illustrate how energy efficiency (y-axis) varies with the output length n_{out} (x-axis). We see that different



(a) XS size models. **(b) S size models.** **(c) M size models.**

Figure 3: Energy efficiency of tested LLMs considering $n_{\text{in}} = 64$ and varying output tokens, per size family (XS, S, M).

LLMs exhibit a substantial spread in E_{eff}. For instance, LLama 3.2 1B is the most efficient LLM, reaching 120.71 at $n_{\text{out}} = 256$ and 51.90 at $n_{\text{out}} = 4096$, whereas Falcon-RW 7B is the least efficient, with energy efficiency values of 18.78 and 2.18 at the same output lengths. While the absolute efficiency values differ across models due to architectural differences, the overall behavior is consistent: each model exhibits a non-linear curve with a distinct peak in efficiency.

To highlight this common trend, we construct an aggregated heatmap in Figure 4 by applying min–max normalization to the efficiency values of each LLM and then averaging across the full set. The resulting 3D visualization reports the energy efficiency (z-axis), bounded between 0 and 1, and illustrates how it varies with different output lengths (x-axis) and input lengths (y-axis). This representation reveals a consistent non-linear pattern: the highest efficiency occurs at $n_{\text{in}} = 64$ and $n_{\text{out}} \in [128, 256]$, whereas the lowest efficiency is observed for shortest output lengths $n_{\text{out}} = 64$ combined with the longest input length $n_{\text{in}} = 4096$. These results suggest the existence of "sweet spots" where LLMs achieve the best trade-off between energy consumption and tokens generated.

Quantitatively, comparing for each model the worst efficient tokens combination, always located at the larger input and shorter output, with the highest energy efficiency configuration case, we observe on average a 33.41× (±4.90) increase in efficiency across models. In Appendix A.2 (see Table 5), we report the θ parameters for our *SweetSpot* model, along with the experimental and the estimated peak efficiency configurations, calculated using Equation 6.

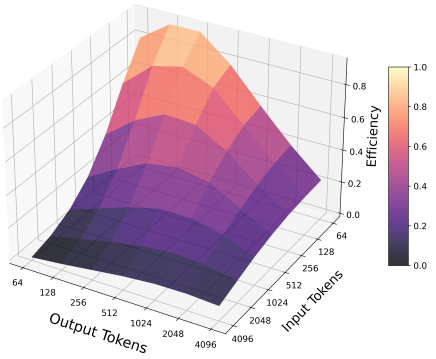


Figure 4: Aggregated energy efficiency heatmap.

5.2.4 ***Prediction of Peak Energy Efficiency Spots.*** To estimate the Energy-per-Token E_{tok} for each combination of n_{in} and n_{out}, we fitted the state-of-the-art Baseline 1-4 models and our proposed analytical model *SweetSpot* described in Table 1 with the results of the experiments of each LLM separately, using a non-linear least square method. In Table 3, we report the Mean Absolute Percentage Error (MAPE) for each model fitted on the relative LLM.

We can observe that the proposed *SweetSpot* model consistently achieves higher accuracy than the baselines for all the tested LLMs. On average, the Baselines 1 and 2 [13] achieve a MAPE of 130.45% and 130.49%, respectively. Baseline 3 [21] improves the MAPE on average to 106.25%, and Baseline 4 [22] further reduces it to 31.40%. This suggests that adding both n_{in} and n_{out} as fitting parameters improves the energy efficiency estimation of LLM inference w.r.t. considering only n_{out}. Our proposed model improves the MAPE over Baseline 4 significantly, reducing it to 2.27% (±0.61) and 1.79% (±0.61) in its FLOPs-only and complete versions respectively. This underlines that the FLOPs-only *SweetSpot* model derived from computational complexity estimates captures better the dependency of LLM inference energy efficiency w.r.t. SoA, and that this dependency is non-linear with both quadratic and mixed terms dependencies. Furthermore, the memory extended *SweetSpot* model consistently reduces the MAPE over the FLOPs-only one with an average reduction of 18.6%, underlining that memory-access operations are as important as FLOPs and the sixth term $\frac{\theta_5}{n_{\text{out}}}$ in the analytical model is crucial.

In particular, the quadratic term n_{in}^2 can be interpreted as the initial cost incurred during the prefill phase. This cost dominates when the input sequence is relatively long, creating a high starting point in the per-token energy curve. As output tokens are generated, the cost reaches an optimal point where the amortization of input costs balances the linear per-token decoding costs. Beyond this optimum, the cost grows linearly with the number of output tokens, reflecting the sequential nature of decoding.

Table 3: Mean Absolute Percentage Error (MAPE) achieved by each analytical and baseline model across the LLMs of the test set.

Model	Baseline 1	Baseline 2	Baseline 3	Baseline 4	SweetSpot (FLOPs-only)	SweetSpot
XS models						
Llama 3.2 (1B)	102.33	99.45	93.58	17.84	2.95	1.56
OPT (1.3B)	139.77	140.98	103.60	42.44	1.29	0.97
Qwen 2 (1.5B)	110.71	100.90	103.60	14.28	2.62	1.94
S models						
Gemma 2 (2B)	111.33	116.29	93.01	25.76	2.17	1.66
OPT (2.7B)	153,29	153.59	111.64	47.14	1.47	0.91
Llama 3.2 (3B)	117.54	121.55	99.77	24.19	2.46	2.12
Granite (3B)	144.28	148.05	107.43	42.44	1.85	1.08
M models						
OPT (6.7B)	151.27	155.47	110.08	46.49	1.82	1.93
Qwen 2 (7B)	132.26	113.34	126.04	15.26	3.24	2.88
Falcon-RW (7B)	150.80	155.75	109.90	47.30	1.48	1.35
Granite (8B)	127.92	128.22	111.45	24.32	2.62	2.24
Llama 3.1 (8B)	123.00	123.41	107.60	23.59	3.01	2.84
Gemma 2 (9B)	131.38	139.41	103.54	37.17	2.49	1.81
Average	**130.45**	**130.49**	**106.25**	**31.40**	**2.27**	**1.79**
Std Dev	**16.23**	**19.14**	**8.22**	**12.21**	**0.61**	**0.61**

Finally, by examining the derivative of the per-token energy with respect to n_{out}, we can identify the points of maximal efficiency observed in the experiments. The derivative analysis confirms the presence of an optimal number of output tokens that minimizes the per-token energy, matching the peak efficiency points observed in the empirical data.

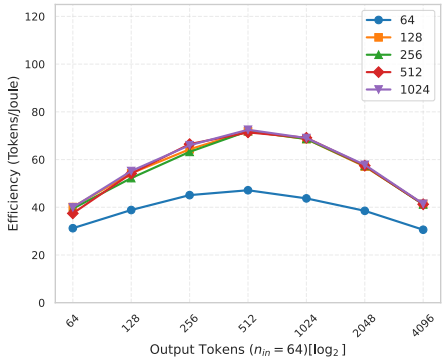
6 LIMITATIONS AND FUTURE WORK

Although the *SweetSpot* model provides meaningful insights into LLM inference energy efficiency, its generality is limited by assumptions on model scale, hardware platform, inference framework, and batching configuration. This section discusses these limitations and outlines directions for future work.

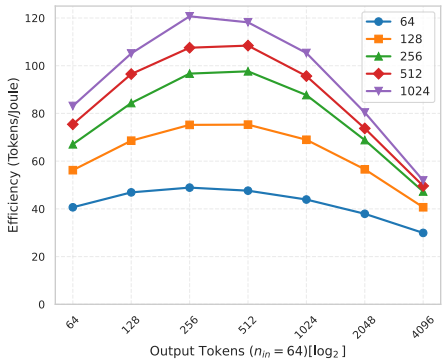
6.1 Experimental Setting

The current study focuses on not very large models, which may not fully capture the performance characteristics of larger LLMs with substantially different parameter counts and memory footprints. Second, all experiments were conducted on a single hardware platform, limiting the generalizability of the results to heterogeneous systems with different compute, memory, and interconnect properties. Finally, we relied on a single inference framework, which constrains the scope of our conclusions, as alternative inference solutions may employ different optimization strategies and execution models affecting performance.

Future work will address these limitations by extending the experimental study to a broader range of model sizes, including large-scale and emerging LLM architectures, and by evaluating performance across multiple hardware platforms. In addition, incorporating multiple inference frameworks will allow a more systematic comparison of software-level optimizations and request scheduling strategies. Developing such a generalized performance model would provide a more comprehensive framework for predicting LLM inference efficiency and for guiding optimization strategies across a wider range of models and deployment scenarios.



(a) $N_{\text{req}} = 128$



(b) $N_{\text{req}} = 1000$

Figure 5: Energy efficiency of Llama 3.2 1B varying maximum batch size considering different numbers of requests.

6.2 Batch Size Impact

The proposed *SweetSpot* analytical model does not explicitly account for the batch size, instead assumes a fixed maximum batch size, which is set to 1024 in all evaluations. As shown in Figure 1, the number of concurrent requests has a strong impact on energy efficiency, particularly at peak values, as the engine can process a larger number of requests simultaneously. To analyze the interaction between the batch size and the number of requests, in Figure 5 we report the energy efficiency of Llama 3.2 1B (the most efficient LLM in the test set) while varying the maximum batch size configuration (`max_batch_size`) across a range of values: 64, 128, 256, 512 and 1024. We tested these configurations for $N_{\text{req}} = 128$ and $N_{\text{req}} = 1000$, with the input sequence length fixed to $n_{\text{in}} = 64$, where the maximum peak of energy efficiency occurs.

In Figure 5a, we observe that setting a maximum batch size larger than $N_{\text{req}} = 128$ does not provide any efficiency benefit. Conversely, a smaller batch size like 64 leads to a significant reduction in efficiency, as this prevents the engine from forming larger batches, resulting in under utilization of computational resources.

When considering a higher number of requests ($N_{\text{req}} = 1000$), as illustrated in Figure 5b, the maximum batch size has a strong effect on overall efficiency. In particular, the peak efficiency decreases from 120.71 with a batch size of 1024 to 48.90 when the batch size is reduced to 64. However, the impact of the batch size diminishes for longer output sequences. For instance, when $n_{\text{out}} = 4096$, the efficiency gap between batch sizes of 256 and 1024 narrows to 47.27 and 51.91, respectively. This reduced sensitivity to batch size is attributable to the decode phase becoming dominant in requests with short inputs and long outputs, causing the engine to be constrained primarily by memory bandwidth rather than by the batch dimension.

In future work, we will extend our analysis of how maximum batch size affects LLM energy efficiency and explicitly incorporate this parameter into the proposed analytical model.

7 CONCLUSIONS

This paper presented a comprehensive study of the energy efficiency of LLM inference evaluated on NVIDIA H100 GPUs with TensorRT-LLM. We showed that energy consumption during inference can be explained by decomposing FLOPs and memory-access contributions into prefill and decode phases, and by normalizing them on a per-token basis. The resulting analytical model, *SweetSpot*, highlights the quadratic overhead of input processing, the amortization effect of output length, and the linear cost of sequential decoding. Our experiments confirm that *SweetSpot* aligns closely with real-world energy measurements. Specifically, we identified efficiency regimes where energy-per-token reaches a minimum, and demonstrated that these correspond to the balance point where input costs are sufficiently amortized without incurring excessive decoding overhead. This explains why efficiency peaks at short-to-moderate input lengths and medium outputs, while dropping significantly for long prompts or short generations.

Beyond theoretical insight, our findings have practical implications for LLM deployment. The analytical structure of inference costs allow to foresee energy efficiency "sweet-spots" configurations and to adapt prompt or generation length to reduce energy consumption. This perspective transforms energy efficiency from an empirical concept into a predictable design parameter. Building a generalized compute–memory efficiency model will further broaden the applicability, supporting sustainable optimization across different architectures and deployment contexts. In summary, our study represents an intermediate step between empirical benchmarking and theoretical modeling of LLM inference costs, providing a model that not only explains current efficiency regimes but also guides more energy-aware deployment strategies for large-scale generative AI.

ACKNOWLEDGMENTS

The research has been partially funded by the EU Pilot for exascale EuroHPC EUPEX (g.a. 101033975), EuroHPC JU SEANERGYS (g.a. 101177590), DARE (g.a. 101143421) and Spoke "FutureHPC & BigData" of the ICSC-Centro Nazionale di Ricerca in "High Performance Computing, Big Data & Quantum Computing", funded by the EU - NextGenerationEU projects.

REFERENCES

[1] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. arXiv:2305.13245 [cs.CL] https://arxiv.org/abs/2305.13245

[2] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, et al. 2023. The Falcon Series of Open Language Models. arXiv:2311.16867 [cs.CL] https://arxiv.org/abs/2311.16867

[3] Reza Yazdani Aminabadi, Samyam Rajbhandari, Minjia Zhang, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Jeff Rasley, Shaden Smith, Olatunji Ruwase, and Yuxiong He. 2022. DeepSpeed Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale. arXiv:2207.00032 [cs.LG] https://arxiv.org/abs/2207.00032

[4] Andrea Bartolini, Francesco Beneventi, Andrea Borghesi, Daniele Cesarini, Antonio Libri, Luca Benini, and Carlo Cavazzoni. 2019. Paving the way toward energy-aware and automated datacentre. In *Workshop Proceedings of the 48th International Conference on Parallel Processing*. "", "", 1–8.

[5] NVIDIA Corporation. 2025. Python Bindings for NVIDIA Management Library (pynvml). https://pypi.org/project/nvidia-ml-py3/. Accessed: 2025-09-12.

[6] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. arXiv:2205.14135 [cs.LG] https://arxiv.org/abs/2205.14135

[7] Jared Fernandez, Clara Na, Vashisth Tiwari, Yonatan Bisk, Sasha Luccioni, and Emma Strubell. 2025. Energy Considerations of Large Language Model Inference and Efficiency Optimizations. arXiv:2504.17674 [cs.CL] https://arxiv.org/abs/2504.17674

[8] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, et al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[9] Nidhal Jegham, Marwan Abdelatti, Lassad Elmoubarki, and Abdeltawab Hendawi. 2025. How Hungry is AI? Benchmarking Energy, Water, and Carbon Footprint of LLM Inference. arXiv:2505.09598 [cs.CY] https://arxiv.org/abs/2505.09598

[10] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. arXiv:2309.06180 [cs.LG] https://arxiv.org/abs/2309.06180

[11] Paul Joe Maliakel, Shashikant Ilager, and Ivona Brandic. 2025. Investigating Energy Efficiency and Performance Trade-offs in LLM Inference Across Tasks and DVFS Settings. arXiv:2501.08219 [cs.LG] https://arxiv.org/abs/2501.08219

[12] Mayank Mishra, Matt Stallone, Gaoyuan Zhang, Yikang Shen, Aditya Prasad, Adriana Meza Soria, Michele Merler, Parameswaran Selvam, et al. 2024. Granite Code Models: A Family of Open Foundation Models for Code Intelligence. arXiv:2405.04324 [cs.AI] https://arxiv.org/abs/2405.04324

[13] Chenxu Niu, Wei Zhang, Yongjian Zhao, and Yong Chen. 2025. Energy Efficient or Exhaustive? Benchmarking Power Consumption of LLM Inference Engines. *SIGENERGY Energy Inform. Rev.* 5, 2 (Aug. 2025), 56–62. https://doi.org/10.1145/3757892.3757900

[14] NVIDIA. 2025. TensorRT-LLM. https://github.com/NVIDIA/TensorRT-LLM. Version 0.21.0, accessed September 15, 2025.

[15] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. ZeRO: Memory Optimizations Toward Training Trillion Parameter Models. arXiv:1910.02054 [cs.LG] https://arxiv.org/abs/1910.02054

[16] Noam Shazeer. 2019. Fast Transformer Decoding: One Write-Head is All You Need. *CoRR* abs/1911.02150 (2019), 0. arXiv:1911.02150 http://arxiv.org/abs/1911.02150

[17] Jovan Stojkovic, Esha Choukse, Chaojie Zhang, Inigo Goiri, and Josep Torrellas. 2024. Towards Greener LLMs: Bringing Energy-Efficiency to the Forefront of LLM Inference. arXiv:2403.20306 [cs.AI] https://arxiv.org/abs/2403.20306

[18] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, et al. 2024. Gemma: Open Models Based on Gemini Research and Technology. arXiv:2403.08295 [cs.CL] https://arxiv.org/abs/2403.08295

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 6000–6010.

[20] Chen Wang, Jialin Qiao, Xiangdong Huang, Shaoxu Song, Haonan Hou, Tian Jiang, Lei Rui, Jianmin Wang, et al. 2025. Apache IoTDB: A Time Series Database for Large Scale IoT Applications. *ACM Trans. Database Syst.* 50, 2, Article 7 (May 2025), 45 pages. https://doi.org/10.1145/3726523

[21] Patrick Wilhelm, Thorsten Wittkopp, and Odej Kao. 2025. Beyond Test-Time Compute Strategies: Advocating Energy-per-Token in LLM Inference. In *Proceedings of the 5th Workshop on Machine Learning and Systems* (World Trade Center, Rotterdam, Netherlands) *(EuroMLSys '25)*. Association for Computing Machinery, New York, NY, USA, 208–215. https://doi.org/10.1145/3721146.3721953

[22] Grant Wilkins, Srinivasan Keshav, and Richard Mortier. 2024. Offline Energy-Optimal LLM Serving: Workload-Based Energy Models for LLM Inference on

Heterogeneous Systems. arXiv:2407.04014 [cs.DC] https://arxiv.org/abs/2407.04014

[23] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, Online, 38–45. https://www.aclweb.org/anthology/2020.emnlp-demos.6

[24] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, et al. 2024. Qwen2 Technical Report. arXiv:2407.10671 [cs.CL] https://arxiv.org/abs/2407.10671

[25] Zhihang Yuan, Yuzhang Shang, Yang Zhou, Zhen Dong, Zhe Zhou, Chenhao Xue, Bingzhe Wu, Zhikai Li, Qingyi Gu, Yong Jae Lee, Yan Yan, Beidi Chen, Guangyu Sun, and Kurt Keutzer. 2024. LLM Inference Unveiled: Survey and Roofline Model Insights. arXiv:2402.16363 [cs.CL] https://arxiv.org/abs/2402.16363

[26] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, et al. 2022. OPT: Open Pre-trained Transformer Language Models. arXiv:2205.01068 [cs.CL] https://arxiv.org/abs/2205.01068

A APPENDIX

This appendix provides additional quantitative details supporting the modeling and statistical analysis presented in the main body of the paper. In particular, it reports the complete set of regression parameter significance statistics and values for our proposed *SweetSpot* analytical model across all evaluated LLMs.

A.1 Parameter Significance

Table 4 summarizes the aggregated statistical significance of each analytical model parameter across all evaluated LLMs. By reporting the distribution of p-values for each parameter, this table highlights which components of the model consistently contribute to explaining performance variation and which parameters exhibit weaker or model-dependent influence. Together, these tables complement the main results by offering a detailed view of the model fitting behavior and its statistical robustness across different architectures and scales. In the table, the number of asterisks quantify the significance of the parameter, with (***) meaning high significance, (**) moderate significance, (*) weaker significance and (n.s.) not significant.

A.2 Parameter Values

Table 5 lists the fitted values of the model parameters θ_0–θ_5 for each LLM, grouped by scale. In addition, it reports both the empirically observed peak energy-efficiency configurations $(n_{\text{in}}, n_{\text{out}})$ and the corresponding values estimated by our *SweetSpot* model, enabling a direct comparison between measured and predicted optimal configurations. These results provide transparency into the parameterization of the model and allow reproducibility of the *SweetSpot* prediction process.

Table 4: Aggregated statistical significance across LLM datasets.

Parameter	$p < 0.001$ (***)	$p < 0.01$ (**)	$p < 0.05$ (*)	$p \geq 0.05$ (n.s.)
θ_0	13/13 (100%)	0/13 (0%)	0/13 (0%)	0/13 (0%)
θ_1	13/13 (100%)	0/13 (0%)	0/13 (0%)	0/13 (0%)
θ_2	13/13 (100%)	0/13 (0%)	0/13 (0%)	0/13 (0%)
θ_3	13/13 (100%)	0/13 (0%)	0/13 (0%)	0/13 (0%)
θ_4	13/13 (100%)	0/13 (0%)	0/13 (0%)	0/13 (0%)
θ_5	8/13 (62%)	1/13 (8%)	3/13 (23%)	1/13 (8%)

Table 5: Theta values and Peak Energy Efficiency Configuration experimented and predicted by our SweetSpot model.

Model	θ_0	θ_1	θ_2	θ_3	θ_4	θ_5	Peak E_{eff} Config	Peak E_{eff} Config (Predicted)
XS models								
Llama 3.2 (1B)	5.005153e-03	1.079941e-07	6.825240e-06	2.611042e-03	3.852659e-06	5.406443e-01	64/256	64/429
OPT (1.3B)	5.969229e-03	1.666292e-07	4.388860e-05	2.915735e-03	2.342971e-05	3.187084e-01	64/64	64/147
Qwen 2 (1.5B)	6.528509e-03	9.147483e-08	6.199954e-06	3.700520e-03	3.288754e-06	3.805096e-01	64/512	64/433
S models								
Gemma 2 (2B)	9.468873e-03	1.223718e-07	2.486662e-05	5.371303e-03	1.372551e-05	5.856132e-01	64/256	64/260
OPT (2.7B)	6.541956e-03	2.855932e-07	9.197051e-05	5.909706e-03	5.038964e-05	8.566784e-01	64/128	64/157
Llama 3.2 (3B)	9.259155e-03	1.842970e-07	2.709305e-05	6.842019e-03	1.461407e-05	8.942031e-01	64/256	64/302
Granite (3B)	9.300161e-03	4.210179e-07	9.340093e-05	7.914036e-03	5.199433e-05	1.170166e+00	64/128	64/180
M models								
OPT (6.7B)	1.300082e-02	4.308046e-07	1.440099e-04	1.379572e-02	8.632801e-05	1.016691e+00	64/128	64/148
Qwen 2 (7B)	1.857219e-02	2.598309e-07	1.531846e-05	1.465957e-02	9.735924e-06	8.719728e-01	64/512	64/431
Falcon-RW (7.5B)	1.871769e-02	9.200243e-07	1.652377e-04	1.673305e-02	1.052294e-04	7.410144e-01	64/128	64/131
Granite (8B)	2.073270e-02	3.926489e-07	4.029094e-05	1.744363e-02	2.512785e-05	8.540071e-01	64/256	64/280
Llama 3.1 (8B)	1.970934e-02	3.329326e-07	3.583505e-05	1.551942e-02	2.249613e-05	8.939092e-01	64/256	64/290
Gemma 2 (9B)	1.749322e-02	5.981696e-07	1.110917e-04	1.901597e-02	7.408055e-05	2.558409e+00	64/64	64/226