Energy Efficiency in LLM Inference: Comparing Inference Libraries in a Unified Docker Framework

Reinier Schep
Delft University of Technology
rschep@tudelft.nl

Razvan Loghin
Delft University of Technology
rloghin@tudelft.nl

Maosheng Jiang
Delft University of Technology
m.jiang-5@student.tudelft.nl

Alex Zheng
Delft University of Technology
azheng@tudelft.nl

April 4, 2025

Abstract

The increasing adoption of Large Language Models (LLMs) has raised concerns about their computational efficiency and energy consumption. This study presents a comparative analysis of four popular LLM inference libraries—Ollama, MLC, vLLM, and TensorRT—evaluating their energy efficiency in a standardized Dockerized environment. Each library is tested for energy consumption in Joules per token generated, and tokens per second are also measured to provide a comprehensive assessment of performance. Our experiments are conducted on identical hardware configurations to ensure fairness, and results indicate significant variations in energy efficiency between frameworks. This work aims to guide researchers and practitioners in selecting the most energy-efficient and performant LLM inference library based on their deployment needs.

Introduction

Cloud computing has become a cornerstone of modern infrastructure, offering on-demand computing resources that power many of today's applications and services. With the rapid growth of Machine Learning (ML) workloads and the proliferation of networked devices, the energy consumption of data centers has risen to alarming levels. Global data centers now consume more energy than most countries, with projections suggesting they could account for up to 20% of global electricity usage by 2025, significantly contributing to global carbon emissions [1].

The rise of Large Language Models (LLMs) has revolutionized domains such as natural language processing and code generation. However, the computational demands of LLMs, particularly during inference, pose significant challenges in terms of energy consumption and resource utilization [2][3]. For instance, GPT-3, with 175 billion parameters, consumed an estimated 1,287 MWh of energy during training [4]. Although training costs are incurred once, inference processes are continuous, serving millions of daily queries and leading to substantial cumulative energy consumption. For example, GPT-3's inference process consumes approximately 0.0003 kWh per query. When scaled to millions of users, this translates to a significant environmental impact [5]. Consequently, optimizing energy efficiency during inference has become critical.

Problem Statement

The deployment of large language models (LLMs) for inference tasks presents a significant challenge in balancing performance and energy efficiency. While various inference libraries have emerged to accelerate LLM processing, their energy consumption and environmental sustainability remain underexplored. Previous studies have extensively evaluated the performance capabilities of popular open-source inference libraries [6]. However, the energy efficiency of open-source inference libraries has not been thoroughly analyzed in past literature. Since the inference of large LLMs is performed continuously at large scale nowadays, increasing the energy awareness of the inference is becoming more important to promote sustainability.

In this work, we aim to address the lack of energy awareness in modern inference libraries for LLMs. Specifically, we propose a streamlined evaluation process to evaluate the energy efficiency of inference libraries. By providing users more insight in the energy efficiency associated with inference libraries, we hope that users will make a more informed choice between different inference libraries and encourage the adoption of energy efficient solutions.

Solution Proposal

This study proposes a comparative analysis of four popular LLM inference libraries, vLLM [7], Ollama [8], MLC [9] and TensorRT-LLM [10] to evaluate their energy efficiency and performance in a Docker containerized environment. These libraries were chosen because of their widespread use and optimization for

accelerating LLM inference tasks.

To achieve a reliable comparison, we will utilize the SWE-bench [11] benchmark dataset for solving real-world GitHub issues. The energy efficiency and performance will be measured using EnergiBridge [12], a cross-platform measurement utility. By collecting metrics from EnergiBridge, we will be able to evaluate the inference libraries by metrics such as tokens per second and energy per token. All experiments are conducted in controlled containerized Docker environments to ensure reproducibility and mimic realistic scenarios, such as cloud deployments.

Contributions

The main contribution of this paper is the insight into the energy consumption of the different inference libraries for LLMs: vLLM, Ollama, MLC and TensorRT-LLM. Specifically, given that each library has to work with the same pre-trained LLM, how much energy does each library consume per token on solving real-world GitHub issues. Besides that, insight is also given in the raw performance measured in tokens per second produced by the inference library.

We have made our code open-source [13], in order to make our obtained results reproducible. Furthermore, we also encourage fellow green energy researchers to use our energy efficiency evaluations, in order to gain more awareness in the area of energy efficiency.

Related work

Recent papers have shown significant improvements in the accuracy of Natural Language Processing (NLP) tasks. State-of-the-art performance was achieved for a translation task from French to English by Bahdanau et al. [14] by leveraging a single neural network. Another paper by Luong et al. [15] improved on this idea by adding additional complexity in the form of attention layers which yielded even better performance. The use of additional complexity in these models tends to come with better performance, as was the case for the models produced by Peters et al. [16]. They utilize "deep contextualized" word representations which enables better performance but require more computational power. As models have grown in complexity, they are usually trained in large datacenters with specialized hardware instead of general purpose servers. Research has shown that most of these datacenters do not derive their energy from carbon-neutral sources [17] which poses significant harm to the environment. While aforementioned research has shown that training models exhibit a positive correlation between performance and energy usage, much less is known about the energy usage of the inference phase of a model. Recent research effort has focused on developing tools that are able to monitor the energy usage of a single prompt to a Large Language Model [18]. The key idea behind this is that LLMs are trained once, but are used potentially indefinitely. So the energy usage of the training phase is bounded, but the energy usage of the inference is unbounded if the model is continuously deployed.

Recent studies further emphasize that the cumulative energy used for serving LLMs can far exceed the one-time training cost. For example, a year of continuous inference may require over 25× the energy needed for training [19]. While early work focused predominantly on training-phase energy footprints, the growing prevalence of LLM deployment has spurred efforts to measure inference efficiency with greater fidelity. Tools such as MELODI, developed by Husom et al. [20], profile per-request energy usage by leveraging NVIDIA's NVML (via `nvidia-smi`) alongside CPU sensors. Similarly, Wilkins et al. utilize the PowerAPI/PyJoules framework to capture low-level power metrics during inference [21]. These methodologies enable researchers to quantify metrics like joules per token and to pinpoint inefficiencies in the inference pipeline. Empirical evaluations by Samsi et al. [22] and Stojkovic et al. [23] have further demonstrated that optimizations such as tuning batch sizes, multi-GPU configurations, GPU frequency scaling, and power capping can yield substantial energy savings. Moreover, comparative studies by Luccioni et al. [19] underscore that task-specific models are significantly more energy-efficient than expansive general-purpose LLMs, highlighting the need for energy-aware deployment strategies.

Background

Large Language Model

Large Language models are AI models that are trained on a large corpus of text so that they can accurately predict the next token given some input sequence of tokens. Tokens are the units of text that an LLM processes. What constitutes a token depends on the tokenization method used by the LLM, it could be a word, part of a word,, or even a single character. For example, the word "unbelievable" might be tokenized as ["un", "believ", "able"], other tokenizations are obviously possible. Once the input is tokenized, each token is converted to a high dimensional vector which encapsulates the meaning of that word. This sequence of tokens which is now a sequence

of vectors, is passed through an attention block followed by a multilayer perceptron block, this process is repeated some number of times and it changes each vector to represent the context it is in [24]. At the final layer, the model's hidden states contain the context for all tokens in the sequence. This context is then used to predict the most likely next token, and the process can be repeated to generate an entire sequence of text, which becomes the output of the LLM.

Quantization

An LLM has weights that are trained according to some training set; the number of parameters can be huge, i.e., the GPT-3 model has 175 billion of them [25]. FP32 (32-bit floating point) and FP16 (16-bit floating points) are the standard formats used in training LLMs, which provide high accuracy but also incur a high computational cost. Quantization is the act of converting high-precision datatypes like FP32 into lower ones, which have fewer bits and thus use less memory and allow faster computations. The art of quantization is to significantly reduce the computational cost needed to run the model while minimizing the performance loss incurred. Due to compatibility constraints, the four tested inference libraries implement three different quantization techniques. Below is the description of each of them.

GGUF is a file format that can be used for storing (quantized) models [26]; it has executors based on GGML, a Tensor library for machine learning (similar to PyTorch and TensorFlow) [27]. Most GGUF quantizations are based on the K-Quants method. It compresses model weights into a 4-bit representation by employing per-channel or per-group scaling factors to preserve the relative magnitudes of the original weights, thereby reducing memory footprint while maintaining acceptable accuracy. However, there are also combined-bit quantization types where only certain weights are quantized. For example, Q4_K_L quantization, a GGUF quantization type, uses 8-bit quantization for the embedding and output weights and 4-bit for the others [28].

Furthermore, 4-bit AWQ (Activation-aware Weight Quantization) leverages activation statistics to more accurately map 32-bit or 16-bit weights into a 4-bit integer format, thereby reducing quantization error and preserving model performance. By incorporating activation information into the quantization process, 4-bit AWQ provides a robust balance between aggressive compression and minimal loss in inference quality.

Moreover, The q4f16_1 quantization approach from the MLC-LLM inference library converts FP16 weights into a 4-bit format while using 16-bit floating point scaling factors to mitigate quantization errors. This scheme achieves a favorable trade-off between model compression and accuracy, making it particularly effective for deployments in memory-constrained environments. This quantization approach is based on the grouping quantization methods, which are discussed in [29] and [30].

Inference Libraries

Given some trained LLM with weights, inference libraries are responsible for loading the model in memory efficiently and to receive some input, compute the output based on the model weights and deliver that back to the user. Inference libraries utilize various optimizations to speed up this process, such as quantization which was described earlier. Some libraries have hardware-specific optimizations; they can restructure the computations optimally to achieve better performance. Pruning is another technique where the library might remove small weights since they likely have small influence on the outcome in order to boost computational speed. They can also fuse layers together sometimes so that multiple operations are performed in one go instead of one by one. A common fuse operation is to combine a matrix multiplication followed by a ReLu function. Normally, the matrix multiplication would be performed first, storing the result in memory and then applying the ReLu function. In a fused operation both the matrix multiplication and ReLu function happen in a single operation, without the need for storage of intermediate results. The following paragraphs explain each of the analyzed inference libraries in detail.

vLLM is a fast and flexible library for LLM inference and serving. Originally developed in the Sky Computing Lab at UC Berkeley, it is now maintained by a community of contributors [31]. It employs architectural innovations such as PagedAttention and continuous batching, along with optimized CUDA kernels—including integration with FlashAttention and FlashInfer [31]. This way, it achieves a 2- to 4-fold performance improvement [32] compared to the baseline FasterTransformer [33] and Orca [34]. It supports multiple quantization formats, including GPTQ, AWQ, INT4, INT8, FP8, and GGUF, while providing seamless integration with popular HuggingFace models and broad hardware compatibility across NVIDIA GPUs, AMD devices, Intel CPUs, IBM Power CPUs, TPUs, and AWS Trainium/Inferentia accelerators. A comparison by Naman (2024) suggests that vLLM shines when the user needs to handle many requests efficiently [35], while a comparative analysis by Aishwarya Goel and Rajdeep Borgohain states that vLLM stands out with its innovative features like PagedAttention and Continuous Batching, which significantly

enhance inference speed and memory efficiency [36].

TensorRT-LLM, the successor of FasterTransformer, is developed and maintained by NVIDIA, compiling LLMs into highly optimized TensorRT engines that are deployed with a Triton Inference Server to leverage in-flight batching, paged KV caching, and multi-GPU or multi-node inference [37]. It supports advanced quantization schemes such as FP8, FP4, INT4 with AWQ, and INT8 formats and offers a flexible API with both an ahead-of-time compiled TensorRT backend and a PyTorch backend for rapid experimentation, making it particularly effective on NVIDIA CUDA GPUs for high-throughput and energy-efficient production deployments. While its superiority with NVIDIA hardware is well-known [36], this is its only hardware compatibility optimization and can pose compatibility problems for many users.

Ollama is a popular inference library that builds upon the `llama.cpp` framework, which implements Meta's LLaMa architecture in efficient C/C++, thereby optimizing the execution of LLM models based on the LLaMa design. Ollama employs a client-server architecture to execute LLM text generation efficiently [38]. In this system, the user interacts with a client while a Go-based server handles backend processing by calling into the highly optimized C/C++ implementation of `llama.cpp` to perform model quantization using GGML/GGUF formats. Additionally, Ollama supports GPU acceleration on both NVIDIA and AMD devices and utilizes the Metal API on Apple platforms, automatically selecting the optimal model based on the execution environment [39], which makes it well-suited for local inference with ease of deployment and robust model management, even though it may not scale as well for high-throughput production settings [35].

MLC-LLM, also known as MLCEngine, is designed as a universal deployment engine that bridges server and local inference by leveraging techniques such as PagedAttention and fused operators to enhance performance [40] [41]. It utilizes machine learning compilation via Apache TVM to generate portable GPU libraries, enabling efficient operation on a wide range of hardware, including NVIDIA CUDA, AMD ROCm, Metal, Android, iOS, WebGPU, Intel Gaudi, and AWS Inferentia [42]. It supports 3-bit and 4-bit group quantization. In a comparison conducted by Rick Zhou, Larme Zhao, Bo Jiang, and Sean Sheng [43], MLC-LLM offers the lowest Time to First Token (TTFT) and maintains high decoding speeds at lower concurrent user loads, though it struggles to remain competitive with other inference libraries as user demand increases.

Methodology

This section outlines the experimental procedures and setup used to evaluate the energy efficiency and performance of the inference libraries. Our methodology follows a systematic workflow that ensures consistency, reproducibility, and fairness across experiments. Figure 1 presents a high-level workflow diagram, detailing the key phases. For each inference library, one experimental round is initiated by launching the Docker container that activates the EnergiBridge tool to measure energy consumption during both the image loading and the LLM inference process. During each run, the experiment logs the energy used, the number of tokens generated, and the duration for generating every response for the prompts from the SWE-bench dataset. The entire experiment repeated 10 times per library while incorporating adequate cooling-off periods. The logged metrics are then forwarded to a postprocessing engine that computes the average tokens per second and energy per token for each library while eliminating outliers.

Docker Image Selection

All inference libraries will be executed inside a Docker container to ensure their environments are as similar as possible. Docker containers impose overhead in system calls, which leads to additional energy usage, but this overhead is the same for each inference library and thus does not affect comparison [44]. Also, Docker containers are industry standard for deploying LLMs at scale as done by Huggingface [45].

The purpose of the docker image is to keep the environment as consistent as possible while minimizing variability. We chose the "python:3.10.16-bookworm" image as a base for each inference library since this was a stable release and compatible with each library. Every docker image only had its necessary packages installed to run the inference library and maintain stability and consistency throughout the experiments.

Inference Libraries

The inference libraries have been chosen primarily based on their popularity and token decoding speed. To explicate, Ollama has been chosen as it is the most popular library on GitHub, with over 135k stars, which serves as a baseline for our experiment. It is essential to mention that Ollama's back-end inference engine is llama.cpp. Furthermore, vLLM, TensorRT-LLM, and MLC are all comparable in token decoding speed and faster than most of their competitors while

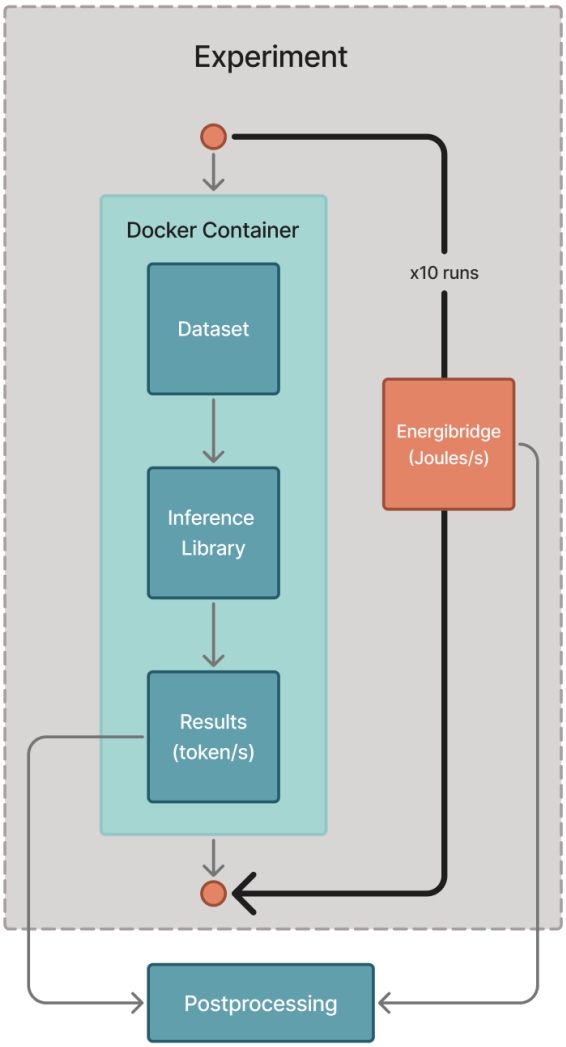

Figure 1: *Methodology workflow diagram*

being popular on GitHub and have therefore been chosen [46].

Text Generation Inference by Hugging Face [47] and LMDeploy [48] are libraries that have not been selected for the experiments but have comparable decoding speeds [49]. An overview of the different inference libraries can be found in Table 1, including the number of GitHub stars each library has.

Table 1: *Overview of different open-source inference libraries compared in experiments.*

Inference Library	Quantization Type	GitHub Stars
Ollama	Q4_K_L	135k
vLLM	4-bit AWQ	43.1k
MLC	q4f16_1	20.3k
TensorRT-LLM	4-bit AWQ	10.1k

Large-Language-Model

Our choice of LLM to experiment with is the quantized to 4-bit version of `Qwen2.5-Coder-14B-Instruct` model with 14B parameters [50]. This LLM is chosen because of its performance and accuracy for coding problems and because it fits within our resource constraints. A quantized model was selected to reduce memory requirements. However, because of compatibility issues, it was not possible to use the identical quantized model for every inference library. Table 1 overviews the exact quantization used with each inference library. The quantization types were chosen based on their best performance and efficiency with comparable VRAM usage on the GPU.

Dataset

The SWE-bench [11] benchmark dataset for solving real-world GitHub issues was chosen for the experiments because it represents a genuine, complex scenario that ensures our findings accurately reflect the energy efficiency and performance of inference libraries in practical settings. This dataset, which has been employed in various previous academic studies [11, 51, 52, 53], was refined for our purposes by selecting 80 entries. This number was determined by the experimental design, where each of the four libraries is evaluated over 10 iterations, while also ensuring that the combined execution time remains within the greenserver's access window.

In addition to the dataset selection, effort was invested in designing the prompt to provide the language model with a robust contextual foundation. The prompt is crafted to contain essential contextual information derived from the repository, such as its identity, the commit history that delineates both the base and environment setup commits, as well as versioning and instance details. It also contains a description of the issue alongside some hints. Although we had the option to include the complete descriptive text of the issue to offer even more context, doing so would have expanded the prompt to a size too large to fit on the available hardware. Nevertheless, we are confident that the constructed prompt contains sufficient detail to facilitate accurate and reliable responses from the language model without supporting hallucinations.

Experimental setup

Every inference library was evaluated through 10 iterations, with each iteration corresponding to a complete pass over the SWE-benchmark dataset of 80 selected prompts. For each new experiment iteration,

the processing order of the libraries was randomly shuffled to mitigate any order bias.

For each run, a dedicated CSV file was created in the respective library's folder to log performance metrics, including the measured energy consumption. A 60-second delay was introduced between each library to ensure system stability and to avoid influence from tail energy usage between libraries.

Moreover, the experiments were conducted using identical parameters across all inference libraries. In particular, we configured the model with the following settings:

- **Context Window:** The total context window (input + output) was set to `32768` tokens. Out of these,
- **Output limit:** `MAX_OUTPUT_TOKENS` was fixed at `4096` tokens, representing the maximum number of output tokens generated.
- **Temperature:** A value of `0.0` was used to achieve as much as possible deterministic results.
- **Seed:** Fixed at `0` to ensure the highest level of determinism in the experimental outcomes.
- **Repetition Penalty:** Set to `1.05` to control repetitive outputs (default).
- **Top-K:** Configured to `20` to reduce the probability of generating incoherent or non-sensical content (default).
- **Top-P:** Set to `0.8` to work in conjunction with the Top-K setting (default).

The experiments were conducted using the GreenServer from the Software Engineering Research Group (SERG) at TU Delft. Below are the key specifications of the machine:

- **CPU:** AMD Ryzen 9 7900X (12x 4.7GHz)
- **Memory:** 64GB DDR5 (2 x 32GB)
- **Storage:** 2TB Kingston Fury Renegade M.2 PCIe 4.0 NVMe SSD
- **Video Card:** NVIDIA RTX4090 24GB

Energy Measurement Process

We have used the EnergiBridge tool to measure the energy used by the GPU and to collect energy-related metrics, such as energy used in Joules. In each docker container, we install the relevant packages and dependencies for the respective inference libraries. These built images are then run on the experimental machine to extract energy efficiency metrics using EnergiBridge.

Furthermore, within a container, we calculate token metrics, such as tokens per second and the total time taken by the inference library. This is then subsequently stored in a CSV file by mounting a volume on the host machine to access the results. For the measurement of energy efficiency, we opt to use the energy per token metric as it is a widely used indicator for measuring energy efficiency for LLMs [54] [55].

Postprocessing

After the results are extracted from the experiment, we remove outliers from the data in order to get a better view of the results, since it is possible that certain runs could produce faulty results due to several reason, for example machine failure and unexpected scenarios. We removed such outliers by removing data instances that differ from the mean by more than 2.9 standard deviations.

Results

Library	Energy/Token (J)		Tokens/Second	
	Mean	Std Dev	Mean	Std Dev
MLC	4.47	0.01	56.49	0.01
Ollama	4.41	0.02	74.46	0.04
vLLM	4.63	0.02	73.85	0.18
TensorRT	3.54	0.01	87.35	0.04

Table 2: *Performance metrics of the four LLM inference libraries*

Table 2 presents the performance metrics for the four inference libraries. In terms of energy efficiency, TensorRT-LLM is the most economical, consuming approximately 3.54 J per token on average, with a low standard deviation (SD) of about 0.01 J, which reflects a high degree of consistency in energy usage. By contrast, both Ollama and MLC require substantially more energy per token, averaging around 4.41 J and 4.47 J respectively, while vLLM exhibits the highest consumption at approximately 4.63 J per token.

Regarding throughput, TensorRT-LLM again demonstrates superior performance by generating roughly 87.35 tokens per second, with minimal variability (SD $\approx$ 0.04). Ollama and vLLM achieve intermediate rates of about 74.46 and 73.85 tokens per second, respectively, whereas MLC lags behind at approximately 56.49 tokens per second. These findings indicate that TensorRT-LLM not only delivers the best energy efficiency but also excels in token generation speed.

Figures 2 and 3 illustrate the distributions of energy consumption per token and tokens per second for the libraries. The data for TensorRT-LLM is notably tightly clustered, underscoring its stable and efficient performance, while vLLM shows greater variability. Both Ollama and MLC maintain moderate consistency, though at higher energy costs.

Overall, the results confirm that TensorRT-LLM outperforms the other libraries in terms of both energy efficiency and throughput. Ollama and vLLM offer competitive token generation speeds at a higher energy expense, and MLC, despite its lower throughput, is the least efficient among the evaluated libraries.

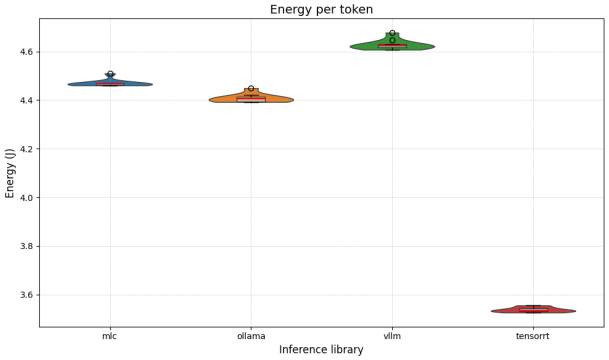

Figure 2: *Energy per token of MLC, Ollama, vLLM, and TensorRT-LLM*

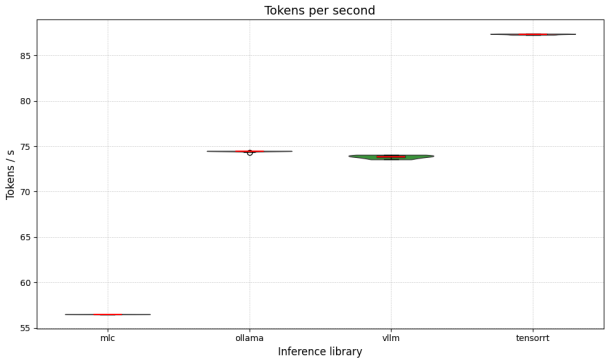

Figure 3: *Tokens per second of MLC, Ollama, vLLM, and TensorRT-LLM*

Discussion

TensorRT-LLM scores both the highest tokens per second and the lowest energy per token. We did not expect a single inference library to score best on both metrics since faster performance usually comes with additional energy costs. However, the performance of TensorRT-LLM could be explained by the fact that it is developed and maintained by NVIDIA, and is optimized for NVIDIA GPUs (a NVIDIA GPU was also used in our experiment). TensorRT-LLM compiles the model into efficient TensorRT engines, which is a key factor in TensorRT-LLM's efficiency. It sweeps through the computational graph of an LLM, selecting the best kernel for each operation based on the available NVIDIA GPU. More crucially, the compiler identifies patterns where multiple operations can be fused into a single kernel [56]. To explicate, the layer fusion technique attempts to fuse different layers of a network together while preserving the behavior, essentially simplifying the network to improve computation speeds [57] and thus reduces the amount of memory movement between the kernels.

Furthermore, in our evaluation, Ollama and vLLM performed very similarly in their token per second metric. Ollama achieves a slightly higher token generation speed and has a lower energy per token than vLLM. vLLM has the highest energy per token, according to our experiment. A potential reason for vLLM's high energy per token could be that the AWQ quantization method is significantly more energy intensive than the K-Quant method of GGUF is [58]. However, TensorRT also uses the AWQ model and is much more energy efficient than both Ollama and vLLM, although TensorRT does have the TensorRT engine optimizations made in its inference engine that vLLM does not integrate. So, despite this advantage in energy efficiency, Ollama was unable to consume less energy per token than TensorRT-LLM. Surprisingly, MLC has the lowest token per second count while consuming energy levels similar to Ollama and vLLM. This was unexpected as MLC also incorporates optimization during the compilation process of the model weights into a custom engine. Yet, MLC natively supports almost any operation platform after compilation.

Moreover, by leveraging quantization, we are able to run bigger models on smaller GPUs since quantization reduces memory requirements. This conversion can be done by the other inference libraries as well, but the difference is that TensorRT also restructures its computations optimally for the NVIDIA GPU hardware. Libraries like MLC, Ollama, and vLLM work across a broader spectrum of hardware (i.e., hardware from AMD) and are thus not optimized for any single platform, which could explain their inferior performance in this experiment. Nonetheless, NVIDIA hardware has a market share of over 90% in the AI supply chain because of their superior performance in ML tasks [59] [60].

Our results suggest that developers should consider energy sustainability when choosing inference libraries. If the user is serving on NVIDIA GPUs, TensorRT-LLM is the preferred inference library. When TensorRT-LLM is not available (e.g., the hardware is not built on NVIDIA GPUs), one could opt for Ollama as the preferred option when generation speed and energy efficiency are the primary requirements.

We further suggest developers of open-source inference libraries to optimize LLM inference libraries for specific computing platforms, rather than developing

cross-platform inference libraries. Ensuring cross-platform compatibility introduces inherent runtime overhead, making it a less optimal solution when good energy efficiency is a desired goal to achieve.

Limitations

Due to compatibility constraints among inference libraries, it was not possible to utilize a single, unified quantization method across all evaluated frameworks. This introduces a bias due to minor variations in the underlying model weights. While our experiment setup accurately represents realistic scenarios—where developers must often navigate quantization compatibility—it slightly complicates direct library comparisons, as differences in performance may partly reflect quantization method efficiency rather than solely library optimizations. However, it is worth noting that inference libraries lacking support for more efficient and/or accurate quantization methods inherently face limitations in achieving optimal efficiency, thus realistically influencing their practical utility.

Another limitation of our research is that we have not experimented with batched inferences, that is, the simultaneous inference of multiple prompts in batches. Our approach feeds the inference libraries one prompt at a time, with a batch size of 1. In a realistic real-world setting, many inference libraries are tasked with inferring prompts in batches to make effective use of resources. This limitation of our work could affect the results, as inference libraries may behave differently when batched inference is utilized instead of inferring single prompts. In addition, with the limited available hardware we had to work with, executing a sizeable batched inference was impossible as memory constraints on the GPU would be broken.

Lastly, an important limitation of our study is that we were unable to evaluate the accuracy of the quantized models and inference libraries against their full-precision base model using metrics such as KL divergence or perplexity (PPL). Due to the GPU memory constraints, we could not run the complete unquantized models, which would have allowed us to quantify the impact of quantization on response quality and accuracy.

Conclusion and Future Work

This research has explored the energy efficiency and performance of different LLM inference libraries: MLC, Ollama, vLLM and TensorRT-LLM. According to the results, TensorRT-LLM performed the best by having both the highest tokens per second and the lowest energy per token. TensorRT-LLM consumed 3538 Joules per token on average and generated 87 tokens per second on average, outperforming all other inference libraries. This indicates that TensorRT-LLM is the fastest inference library for NVIDIA hardware while consuming the least energy.

The research area of green energy and energy efficiency has not yet been widely explored in the context of LLMs, and this enables interesting opportunities for future work. In this work, we have shown a quantitative analysis of energy efficiency and performance of four popular inference libraries for LLMs. Future research that could be built upon our work could explore the energy efficiency of different hardware, for example, hardware from another mainstream GPU manufacturer, Advanced Micro Devices (AMD). One must take into account that TensorRT-LLM will not work in the case of AMD hardware, since it is developed explicitly for NVIDIA hardware. It would be interesting to explore which inference libraries perform best on hardware that is not from NVIDIA.

Since the development of LLMs and inference libraries is at a rapid pace, it also means that more LLMs and inference libraries will be released in the future, possibly with better efficiency and performance. Thus, it is also interesting to benchmark other inference libraries, such as TGI [61], SGLang [62], and LMDeploy [48].

Future research could also explore how different model quantizations in different inference libraries impact the quality or accuracy of LLM output compared to the base model. Investigating whether the subtle trade-offs in quantization and performance translate to differences in output reliability or fidelity is crucial in selecting inference libraries for diverse deployment needs.

Lastly, another related and interesting topic to explore in future work is the energy efficiency and performance of including general knowledge questions in the dataset. In our work, we have exclusively focused on the inference of coding-related prompts, specifically from the SWE-bench of real-world GitHub issues. It would be interesting to explore another common use case of LLMs, answering general knowledge questions.

References

[1] Rajkumar Buyya, Shashikant Ilager, and Patricia Arroba. *Energy-Efficiency and Sustainability in New Generation Cloud Computing: A Vision and Directions for Integrated Management of Data Centre Resources and Workloads*. 2023. arXiv: 2303.10572 [cs.DC]. URL: https://arxiv.org/abs/2303.10572.

[2] Negar Alizadeh et al. "Analyzing the Energy and Accuracy of LLMs in Software Development". In: *arXiv e-prints* (2024), arXiv–2412.

[3] Holistic AI Team. *AI and ESG: Understanding the Environmental Impact of AI and LLMs*. Accessed: 2025-03-19. Mar. 2024. URL: https://www.holisticai.com/blog/environmental-impact-ai-llms.

[4] David Patterson et al. "Carbon emissions and large neural network training". In: *arXiv preprint arXiv:2104.10350* (2021).

[5] Joseph McDonald et al. "Great power, great responsibility: Recommendations for reducing energy for training language models". In: *arXiv preprint arXiv:2205.09646* (2022).

[6] Zain ul Abideen. *Best LLM Inference Engine? TensorRT vs vLLM vs LMDeploy vs MLC-LLM*. Accessed: 2025-03-19. July 2024. URL: https://medium.com/@zaiinn440/best-llm-inference-engine-tensorrt-vs-vllm-vs-lmdeploy-vs-mlc-llm-e8ff033d7615.

[7] Woosuk Kwon et al. "Efficient Memory Management for Large Language Model Serving with PagedAttention". In: *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*. 2023.

[8] Ollama Developers. *Ollama: Local Large Language Model Framework*. https://github.com/ollama/ollama. Accessed: 2025-03-30. 2025.

[9] MLC team. *MLC-LLM*. 2023-2025. URL: https://github.com/mlc-ai/mlc-llm.

[10] NVIDIA Corporation. *TensorRT-LLM: A library for optimizing large language model inference on NVIDIA GPUs*. 2023. URL: https://github.com/NVIDIA/TensorRT-LLM.

[11] Carlos E. Jimenez et al. *SWE-bench: Can Language Models Resolve Real-World GitHub Issues?* 2024. arXiv: 2310.06770 [cs.CL]. URL: https://arxiv.org/abs/2310.06770.

[12] June Sallou, Luís Cruz, and Thomas Durieux. *EnergiBridge: Empowering Software Sustainability through Cross-Platform Energy Measurement*. 2023. arXiv: 2312.13897 [cs.SE]. URL: https://arxiv.org/abs/2312.13897.

[13] Reinier Schep. *CS4575 Project 2*. https://github.com/flazedd/cs4575-project2. Accessed: 2025-04-01. 2024.

[14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL]. URL: https://arxiv.org/abs/1409.0473.

[15] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. *Effective Approaches to Attention-based Neural Machine Translation*. 2015. arXiv: 1508.04025 [cs.CL]. URL: https://arxiv.org/abs/1508.04025.

[16] Matthew E. Peters et al. *Deep contextualized word representations*. 2018. arXiv: 1802.05365 [cs.CL]. URL: https://arxiv.org/abs/1802.05365.

[17] Emma Strubell, Ananya Ganesh, and Andrew McCallum. "Energy and Policy Considerations for Deep Learning in NLP". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Ed. by Anna Korhonen, David Traum, and Lluís Màrquez. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 3645–3650. DOI: 10.18653/v1/P19-1355. URL: https://aclanthology.org/P19-1355/.

[18] Erik Johannes Husom et al. *The Price of Prompting: Profiling Energy Use in Large Language Models Inference*. 2024. arXiv: 2407.16893 [cs.CY]. URL: https://arxiv.org/abs/2407.16893.

[19] Alexandra Sasha Luccioni, Yacine Jernite, and Emma Strubell. "Power Hungry Processing: Watts Driving the Cost of AI Deployment?" In: *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency (FAccT)*. 2024.

[20] Erik Johannes Husom et al. "The Price of Prompting: Profiling Energy Use in Large Language Models Inference". In: *arXiv preprint arXiv:2407.16893* (2024).

[21] PowerAPI Team. *PyJoules: Python-based energy measurement library for various domains including NVIDIA GPUs*. https://github.com/powerapi-ng/pyJoules. Accessed: 2024-01-10. 2024.

[22] Siddharth Samsi et al. "From Words to Watts: Benchmarking the Energy Costs of Large Language Model Inference". In: *Proceedings of the 2023 IEEE High Performance Extreme Computing Conference (HPEC)*. 2023, pp. 1–9. DOI: 10.1109/HPEC58863.2023.10363447.

[23] Jovan Stojkovic et al. "Towards Greener LLMs: Bringing Energy-Efficiency to the Forefront of LLM Inference". In: *arXiv preprint arXiv:2403.20306* (2024).

[24] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[25] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL]. URL: https://arxiv.org/abs/2005.14165.

[26] Gerganov. *ggml/docs/gguf.md at master · ggml-org/ggml — github.com*. https://github.com/ggml-org/ggml/blob/master/docs/gguf.md. [Accessed 04-04-2025].

[27] Xuan-Son Nguyen, Georgi Gerganov, and Slaren. *Introduction to ggml*. Aug. 2024. URL: https://huggingface.co/blog/introduction-to-ggml.

[28] Arnav Chavan et al. "Faster and lighter llms: A survey on current challenges and way forward". In: *arXiv preprint arXiv:2402.01799* (2024).

[29] Tim Dettmers and Luke Zettlemoyer. *The case for 4-bit precision: k-bit Inference Scaling Laws*. 2023. arXiv: 2212.09720 [cs.LG]. URL: https://arxiv.org/abs/2212.09720.

[30] Gunho Park et al. "Lut-gemm: Quantized matrix multiplication based on luts for efficient inference in large-scale generative language models". In: *arXiv preprint arXiv:2206.09557* (2022).

[31] vLLM Team. *vLLM Documentation*. vLLM. 2023. URL: https://docs.vllm.ai/en/latest/.

[32] Woosuk Kwon et al. *Efficient Memory Management for Large Language Model Serving with PagedAttention*. 2023. arXiv: 2309.06180 [cs.LG]. URL: https://arxiv.org/abs/2309.06180.

[33] NVIDIA. *FasterTransformer*. Accessed: 2024-03-29. 2024. URL: https://github.com/NVIDIA/FasterTransformer.

[34] Gyeong-In Yu et al. "Orca: A Distributed Serving System for Transformer-Based Generative Models". In: *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, 2022, pp. 521–538. URL: https://www.usenix.org/conference/osdi22/presentation/yu.

[35] Naman. *Ollama vs vLLM: Which Tool Handles AI Models Better?* 2024. URL: https://medium.com/@naman1011/ollama-vs-vllm-which-tool-handles-ai-models-better-a93345b911e6.

[36] Rajdeep Borgohain Aishwarya Goel. *vLLM vs TensorRT-LLM: Which Inference Library is Best for Your LLM Needs?* 2024. URL: https://www.inferless.com/learn/vllm-vs-tensorrt-llm-which-inference-library-is-best-for-your-llm-needs.

[37] NVIDIA. *TensorRT-LLM*. Accessed: 2024-03-29. 2024. URL: https://github.com/NVIDIA/TensorRT-LLM.

[38] Rife Wang. *Analysis of Ollama Architecture and Conversation Processing Flow for AI LLM Tool*. Medium. 2024. URL: https://medium.com/@rifewang/analysis-of-ollama-architecture-and-conversation-processing-flow-for-ai-llm-tool-ead4b9f40975.

[39] Ollama Team. *Ollama: Run large language models locally*. Version v0.1.33. 2023. URL: https://github.com/ollama/ollama.

[40] MLC AI Team. *Universal LLM Deployment Engine with ML Compilation*. June 7, 2024. URL: https://blog.mlc.ai/2024/06/07/universal-LLM-deployment-engine-with-ML-compilation.

[41] Arnav Chavan et al. *Faster and Lighter LLMs: A Survey on Current Challenges and Way Forward*. 2024. arXiv: 2402.01799 [cs.LG]. URL: https://arxiv.org/abs/2402.01799.

[42] *MLC-LLM GitHub Repository*. MLC AI. 2023. URL: https://github.com/mlc-ai/mlc-llm (visited on 07/18/2024).

[43] *Benchmarking LLM Inference Backends*. BentoML. 2023. URL: https://www.bentoml.com/blog/benchmarking-llm-inference-backends (visited on 07/18/2024).

[44] Eddie Antonio Santos et al. "How does docker affect energy consumption? Evaluating workloads in and out of Docker containers". In: *Journal of Systems and Software* 146 (2018), pp. 14–25. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2018.07.077. URL: https://www.sciencedirect.com/science/article/pii/S0164121218301456.

[45] Docker. *Docker and Hugging Face Partner to Democratize AI*. Accessed: 2024-03-29. 2024. URL: https://www.docker.com/blog/docker-and-hugging-face-partner-to-democratize-ai/.

[46] Rick Zhou et al. *Benchmarking LLM Inference Backends*. 2024. URL: https://www.bentoml.com/blog/benchmarking-llm-inference-backends.

[47] Oct. 2023. URL: https://github.com/huggingface/text-generation-inference.

[48] LMDeploy Contributors. *LMDeploy: A Toolkit for Compressing, Deploying, and Serving LLM*. https://github.com/InternLM/lmdeploy. 2023.

[49] Aishwarya Goel. *Exploring LLMs Speed Benchmarks: Independent Analysis*. 2024. URL: https://www.inferless.com/learn/exploring-llms-speed-benchmarks-independent-analysis.

[50] An Yang et al. "Qwen2 Technical Report". In: *arXiv preprint arXiv:2407.10671* (2024).

[51] John Yang et al. *SWE-bench Multimodal: Do AI Systems Generalize to Visual Software Domains?* 2024. arXiv: 2410.03859 [cs.CL]. URL: https://arxiv.org/abs/2410.03859.

[52] You Wang, Michael Pradel, and Zhongxin Liu. *Are "Solved Issues" in SWE-bench Really Solved Correctly? An Empirical Study*. 2025. arXiv: 2503.15223 [cs.SE]. URL: https://arxiv.org/abs/2503.15223.

[53] Reem Aleithan et al. *SWE-Bench+: Enhanced Coding Benchmark for LLMs*. 2024. arXiv: 2410.06992 [cs.SE]. URL: https://arxiv.org/abs/2410.06992.

[54] Sebastian Bast et al. "LLMs on the Edge: Quality, Latency, and Energy Efficiency". In: *INFORMATIK 2024*. Bonn: Gesellschaft für Informatik e.V., 2024, pp. 1183–1192. ISBN: 978-3-88579-746-3. DOI: 10.18420/inf2024_104.

[55] Grant Wilkins, Srinivasan Keshav, and Richard Mortier. *Hybrid Heterogeneous Clusters Can Lower the Energy Consumption of LLM Inference Workloads*. 2024. arXiv: 2407.00010 [cs.DC]. URL: https://arxiv.org/abs/2407.00010.

[56] *Model Definition*. NVIDIA. 2024. URL: https://nvidia.github.io/TensorRT-LLM/architecture/core-concepts.html.

[57] NVIDIA Corporation. *Best Practices For TensorRT Performance*. Accessed: 2025-03-30. 2024. URL: https://docs.nvidia.com/deeplearning/tensorrt/archives/tensorrt-803/best-practices/index.html.

[58] Saurabhsingh Rajput and Tushar Sharma. *Benchmarking Emerging Deep Learning Quantization Methods for Energy Efficiency*. 2024. DOI: 10.1109/ICSA-C63560.2024.00049.

[59] Steven Farrell et al. "MLPerf™ HPC: A holistic benchmark suite for scientific machine learning on HPC systems". In: *2021 IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (MLHPC)*. IEEE. 2021, pp. 33–45.

[60] Mark Bergen. *Are AI Monopolies Here to Stay? Nvidia and the Future of AI Chips*. Mar. 2025. URL: https://www.bloomberg.com/news/features/2025-03-20/are-ai-monopolies-here-to-stay-nvidia-and-the-future-of-ai-chips.

[61] Hugging Face. *text-generation-inference*. Accessed: 2025-04-04. 2025. URL: https://github.com/huggingface/text-generation-inference.

[62] Lianmin Zheng et al. "Sglang: Efficient execution of structured language model programs". In: *Advances in Neural Information Processing Systems* 37 (2024), pp. 62557–62583.