EchoSwift

An Inference Benchmarking and Configuration Discovery Tool for Large Language Models (LLMs)

Karthik Krishna
CTO
InfobellIT Solutions Pvt. Ltd
Bengaluru, Karnataka, India
karthik@infobellit.com

Ramana Bandili
CEO
InfobellIT Solutions Pvt. Ltd
Bengaluru, Karnataka, India
braman@infobellit.com

ABSTRACT

Large Language Models (LLMs) are advanced natural language processing models that are trained on vast amounts of text data to understand and generate human-like language. These models are designed to understand context, generate coherent and contextually relevant text, and demonstrate advanced language capabilities. In the dynamic landscape of LLMs, the demand for efficient inference benchmarking is crucial.

Organizations such as TPC and SPEC brought several industry standard benchmarks [1][2][3][4]. This publication introduces EchoSwift [11], a comprehensive benchmarking framework designed to evaluate the real-time performance of LLMs in deployment scenarios.

As LLMs ascend to the forefront of technological innovation, their seamless integration into real-world applications demands a nuanced understanding of their efficiency, throughput, latency, and scalability. It is within this dynamic landscape that our publication unveils the EchoSwift, a novel benchmarking framework meticulously crafted to address the pressing need for comprehensive inference benchmarking, as well as the discovery of the right configuration for specific LLM requirements. For instance, certain deployments might have 32 tokens as input and 256 tokens as output, while others might have 256 tokens as input and 64 tokens as output. It is crucial to acknowledge that the configuration for these two requirements need not be the same for an optimal performance, scale and better TCO. The EchoSwift not only aids in comprehensive configuration discovery but also facilitates robust Performance/Scale testing, ensuring that LLM deployments are not only efficient but also finely tuned to their specific operational demands.

*Both authors contributed equally to this research.

ICPE Companion '24 , May 7–11, 2024, London, United Kingdom

ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3652273

CSS CONCEPTS

• **Computer Systems Organization → Artificial Intelligence → Natural Language Processing.**

KEYWORDS

Large language models, Text generation Inference, Llama2, LLM Performance, AI Benchmarking

ACM Reference format:

Karthik Krishna and Ramana Bandili. 2024. EchoSwift: An Inference Benchmarking and Configuration Discovery Tool for Large Language Models (LLMs), 2024. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering. May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA. https://doi.org/10.1145/3629527.3652273

1. INTRODUCTION

LLMs have become so profound that language comprehension and production have transcended traditional boundaries, making it imperative to gauge the real-time performance of these models in deployment scenarios more crucial than ever. The advent of LLMs, exemplified by models like the Llama2 from Meta with varying parameters and precision levels, has propelled them into the core of applications ranging from natural language processing to AI-driven services.

This publication delves into the intricate challenges posed by diverse LLM variants. Llama2 is one such open sourced publicly available LLM and this benchmarking tool was primarily tested with Llama2, however, this tool is applicable to all different LLMs deployed with various architectures and technologies. Llama2 is an advanced AI platform that combines cutting-edge algorithms, extensive data sets, and powerful computational capabilities to deliver exceptional results. Llama2 model has various models which different in parameters such as 7B, 13B, and 70B, coupled with precision nuances in BF16, Int8, and Int4. These intricacies make the identification of an ideal and efficient infrastructure for serving these models a formidable challenge. Enter EchoSwift – a compass guiding practitioners through the delicate balance between model complexity and operational efficiency in the realm of LLMs.

In this publication, we embark on a journey to introduce and expound upon EchoSwift, a benchmarking framework tailored to

assess the real-time performance of LLMs. As we traverse through the subsequent sections, we unravel the significance of this framework, its methodology, and the pivotal role it plays in shaping the deployment landscape for Large Language Models.

2. BACKGROUND

Before the advent of LLMs, a substantial 70% of the AI Inference market was dominated by CPU architectures, highlighting the transformative shift brought about by the introduction of LLMs in the landscape of inference processing. Within the burgeoning landscape of LLMs, this publication unveils EchoSwift – a pioneering benchmarking framework meticulously crafted to assess the real-time performance of LLMs in deployment scenarios. The results presented here reflect out-of-the-box performance with currently released software, with the anticipation of additional performance gains in upcoming releases.

3. ECHOSWIFT OVERVIEW APPROACH

The article outlines benchmarking the performance of LLM using LLama2-7B as the sample LLM model and measures Token Latency, Throughput calculated as tokens per second, and Time To First Token (TTFT).

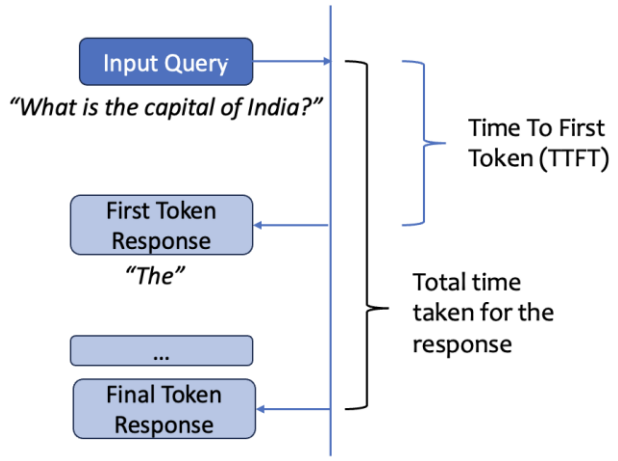

Figure 1: Performance Metrics

Latency is measured of time to output each token when streaming the output excluding the first token and is often measured in millisecond.

$$Latency = \frac{Total\ time\ to\ output - TTFT}{Total\ Number\ of\ Tokens - 1}$$

TTFT is the time to process the prompt and output the first token and is often measured in millisecond.

$$TTFT = Time\ To\ First\ Token$$

Throughput is calculated as tokens per second which takes in to account the total time taken to output all the tokens and normalized to 1 second, i.e., total tokens for the output divided by total time taken in seconds.

$$Throughput = \frac{Total\ No\ of\ tokens}{Total\ Time\ to\ Output\ (in\ seconds)}$$

The EchoSwift Benchmark is used in two modes:

a) Configuration Discovery Mode

b) Performance Benchmarking Mode

In Configuration Discovery mode, we restrict the number of parallel requests to 1, 3, or 10 while varying the parameters for input token size and output token size based on the specific requirements of the application. We employ this approach to test different scenarios and identify Token Latency, Throughput, and TTFT for various combinations of Input and Output tokens. The data obtained is then used to discover the optimal configuration.

In Performance Benchmarking Mode, we maintain the input token and output token size as constants (for example, 32 tokens for Input and 256 tokens for output or any other combination specific to the application requirement) and scale the number of parallel requests (or parallel users) for this fixed combination of a single input and output token. This scaling enables a better sizing of the environment.

4. BENCHMARKING METHODOLOGY AND STEPS

The steps for benchmarking LLM have been discussed below:

4.1. Data Collection

The Hugging Face Hub consists of the vast amount datasets for variety of domains and tasks. The datasets available on Hugging Face is continually expanding, and new datasets are consistently being added by both the Hugging Face team and the community. The Hugging Face Hub hosts a large number of community-curated datasets for a diverse range of domains, languages, and tasks such as translation, automatic speech recognition, and image classification.

Latency per token = (latency – TTFT) / (output tokens -1)

request	start_time	end_time	input_tokens	output_tokens	latency(ms)	throughput(tokens/second)	time_per_token(ms/tokens)	TTFT(ms)
1	1970-01-08 17:25:27.448310	1970-01-08 17:25:37.612477	131	66	10164.167228969745	19.381814128216405	147.5228992004234	575.178780942224
2	1970-01-08 17:25:37.638074	1970-01-08 17:25:44.336981	130	40	6698.907856014557	25.37727099013117	151.86879279700895	776.0249369312078
3	1970-01-08 17:25:44.338007	1970-01-08 17:25:54.837021	127	66	10499.014172004536	18.382678300847676	149.71888535297833	767.2866240609437
4	1970-01-08 17:25:54.838166	1970-01-08 17:26:05.026627	125	66	10188.46081092488	18.746698205404545	147.41419543010684	606.5381079679355
5	1970-01-08 17:26:05.027506	1970-01-08 17:26:15.215437	131	66	10187.930912012234	19.336605410989208	146.22517707757652	683.2944019697607
1	1970-01-08 17:26:27.510252	1970-01-08 17:26:43.019906	132	94	15509.653392946348	14.571569994129126	160.0505927632693	624.9482659623027
2	1970-01-08 17:26:43.041832	1970-01-08 17:27:03.140747	126	130	20098.91493699979	12.737005992733142	150.68639124032515	660.370466997847
3	1970-01-08 17:27:03.141985	1970-01-08 17:27:22.713432	122	130	19571.4472719701	12.87590010580925	147.0961459531528	596.044444013387
4	1970-01-08 17:27:22.714595	1970-01-08 17:27:43.008450	127	130	20293.855173047632	12.663931904930658	152.5422360237269	615.9067259868607
5	1970-01-08 17:27:43.010358	1970-01-08 17:28:03.043577	127	129	20033.219030010514	12.778775074365353	151.77725606190506	605.7302540866658
1	1970-01-08 17:28:25.104480	1970-01-08 17:29:05.333811	127	258	40229.331478942186	9.570131688654236	154.1136843813692	622.114592930302
2	1970-01-08 17:29:05.355198	1970-01-08 17:29:45.507219	124	258	40152.020766050555	9.513842459530448	153.54457519837956	691.0649400670081
3	1970-01-08 17:29:45.508864	1970-01-08 17:30:25.934042	128	258	40425.17778498586	9.548504698063754	154.579172443584	698.3304669847712
4	1970-01-08 17:30:25.936447	1970-01-08 17:31:06.918849	122	258	40982.402284047566	9.272272458950395	157.11764341622504	603.167926077731
5	1970-01-08 17:31:06.920197	1970-01-08 17:31:43.705990	122	243	36785.79278790858	9.922308922480932	149.59577474337118	583.6153000127524

Figure 2: Sample Output for varying combinations of input and output token for Single User

The dataset used here in the benchmark is from ShareGPT Dataset from Hugging Face. Dataset has been filtered based on varying input token lengths. Considered input token lengths in this context range from 32 to 2,000, with variations of approximately ±10 tokens for each length. The specified lengths include 32, 64, 128, 256, 512, 1K and 2K tokens, providing a comprehensive coverage of input sizes for benchmarking. The dataset contains the 7 different files that have 1000 prompts for each token length as specified. Python file DatasetFiltering.py has been used here for Data Processing.

4.2. Configuration Discovery Test

The objective of the work involves identifying the optimal configuration with a single container test.

The analysis involves determining the optimal latency, throughput and TTFT by sending individual requests one at a time with

different input and output tokens. To enhance throughput, parallel requests are then dispatched to the endpoint. Figure 2 above depicts the sample output capturing the performance metrics when a single request for 128 input tokens and with varying output token combinations 64, 128, 256 is given as input request. Similarly varying combinations of input and output tokens in different combinations like 128 output tokens for 128 input and 256 output tokens for 128 inputs for 5 parallel requests are sent to capture the ideal performance parameters.

The maximum throughput is identified when the model consistently provides prompt responses to input requests without significant degradation in latency. This approach allows for a balanced assessment, ensuring that the system achieves optimal performance by striking the right balance between response time and concurrent processing capabilities.

4.3. Scale Testing/Parallel Requests

Locust Load testing has been used for benchmarking setup. Locust is an opensource load testing tool, written in Python and is a highly valuable tool for identifying performance bottlenecks, testing the scalability of system, and ensuring that the developed web applications can handle a specified level of traffic. The tool allows to set the Number of Users which indicates the maximum no. of users that can run simultaneously, and Spawn Rate denotes the number of users that will be spawned per second.

For deployment, hugging face text-generation inference model server 1.1.1 is used.

The steps below need to be followed to run the load test:

1. Define the configurations to run the load test.
2. Listing the parallel users (1, 3, 10, 30) and the Input tokens (32, 64, 128, 256, 512) and Output tokens (32, 64, 128, 256, 512).
3. TGI endpoint has been used for hosting the model.

In Section 5 the results are discussed and the generated graphs for performance metrics have been explained in detail.

5. RESULT ANALYSIS

The result analysis involves determining the optimal latency, throughput and TTFT by sending individual requests one at a time with different input and output tokens. To enhance throughput, parallel requests are then dispatched to the endpoint. This section gives the detailed observation for Configuration Discovery Result analysis and performance test.

5.1. Configuration Discovery Result Analysis

To identify an optimal Configuration to achieve ideal token latency and throughput, the systems are tested with various combinations of input and output tokens. The below graphs

illustrate the Throughput, Token latency and TTFT for single user sending the requests to the endpoint for 32, 64, 128 input tokens and 64,128, 256 output tokens.

In Figure 3, it can be observed that the throughput varies between 4.94 tokens/second to 11.88 tokens/second for single user.

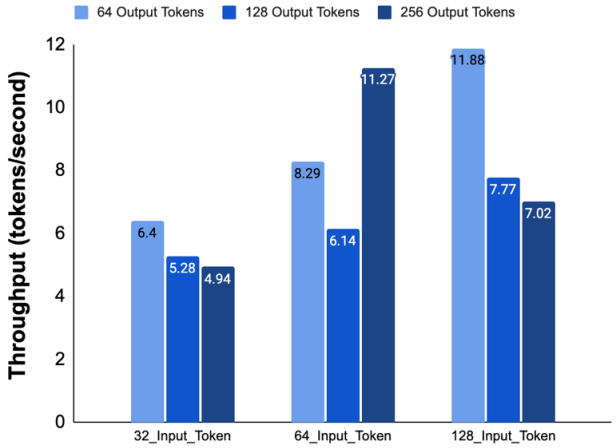

Figure 3: Throughput for Single User

Figure 4 depicts tokens latency for single user ranging from 233 milliseconds/token to 247 milliseconds/token.

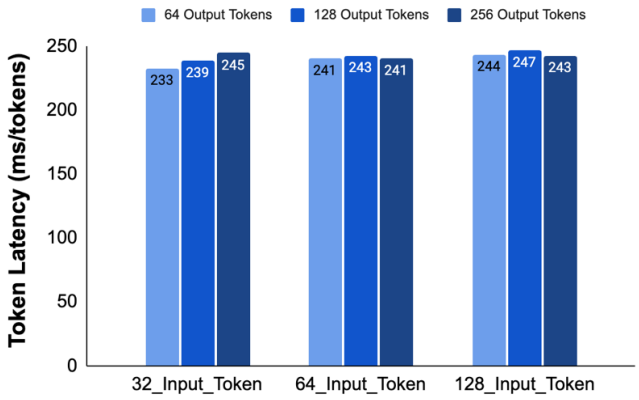

Figure 4: Token Latency for Single User

Similarly, Figure 5 depicts the TTFT for single user it varies between 363 milliseconds to 859 milliseconds.

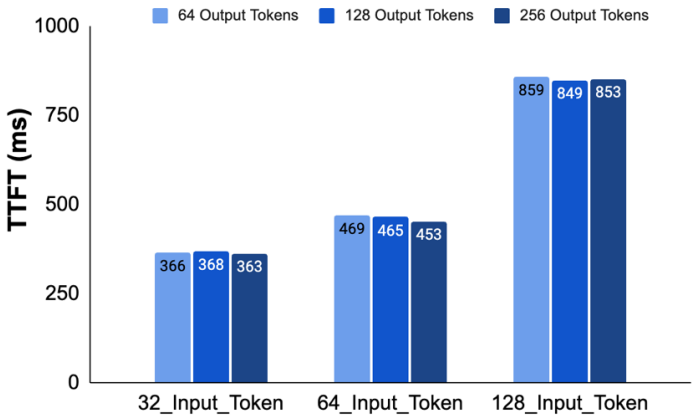

Figure 5: TTFT for a Single User

The achieved performance results were obtained through testing the model on a hardware configuration featuring a 16-core CPU and 128 GB of RAM. It is anticipated that conducting the same load testing on more robust hardware configurations will likely yield even more substantial improvements in performance.

5.2. Performance Test (with Parallel Requests) Result Analysis

The model can also be tested against multiple users for parallel requests sent to the model endpoint for varying input and output tokens combinations. Performance testing with parallel requests is a critical aspect of evaluating the robustness and scalability of a system. When analysing the results of such tests, it is important to consider various factors to gain insights into the system's behaviour under intense loads. Therefore, a comprehensive analysis of performance test is done by examining throughput and token latency against parallel requests sent to the model to get some insights for improvement.

Line graph shown in Figure 6 depicts the relationship between the number of parallel requests made to the model endpoint and the average latency of those requests. It can be observed that when number of parallel requests increases, the average latency also increases due to limited system resources. Thus, it is utmost important to identify the ideal configuration that can handle multiple parallel requests for scale testing.

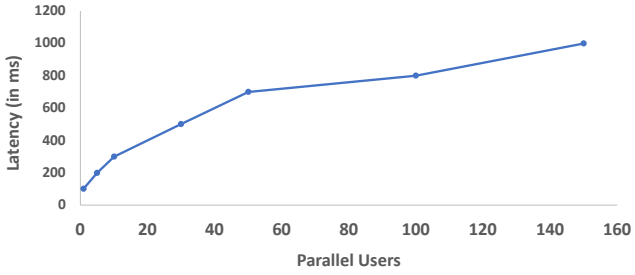

Figure 6: Latency vs Parallel Requests

Graph in Figure 7 shows the relationship between the number of parallel requests made to the model endpoint and the throughput of the system, measured in tokens per second. It can be inferred from the graph that the throughput increases linearly with the requests initially and starts slowing down as the resources become saturated, and eventually decreases when the system is overloaded as the system has limited resources.

Thus, the specific curve and values will vary depending on the specific system and workload, but the general trend is consistent.

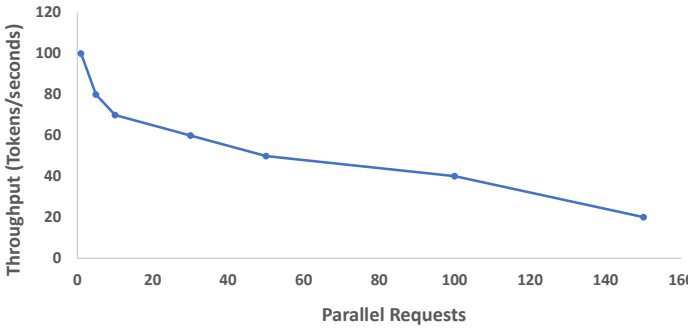

Figure 7: Throughput vs Parallel Requests

The above graphs can be used to understand the performance limitations of a system under increasing load. It can help the users to determine the optimal number of system configurations required to handle the concurrent requests to while maintaining acceptable throughput and latency. Additionally, it can be used to compare the performance of different systems or to track changes in performance over time.

6. CONCLUSION

This benchmark can be used to evaluate a single container, or a cluster with thousands or nodes deploying an LLM. This can be used to test scale, test latency, throughput and TTFT for any environment deploying an LLM. This is not limited to Llama2 but any form of LLM, quantized models with lower precisions (int8, int4, etc) and different precision and different sizes with and without CPU, GPU, Accelerators, or other technology.

This could also be used for inference benchmarking with Retrieval Augmented Generation (RAG) based applications, Fine Tuning models or Fully trained LLM models.

Benchmarking LLMs provides valuable insights for businesses aiming to deploy natural language processing applications. To make the best decisions, it's crucial to acknowledge the specific needs of each application and understand how well LLMs perform on different types of CPUs, GPUs and Accelerators to identify the ideal throughput, latency and scale and drive the total cost of ownership (TCO) lower. Consideration of the specific requirements of each application, coupled with an understanding of the strengths and weaknesses of LLMs on different software and hardware technologies, and architectures empowers businesses to make informed and optimised decisions.

In line with our commitment to standardization and industry best practices, we propose this workload to industry standard organizations like SPEC to create standards for Inference on Large Language Models. Establishing such standards will further facilitate benchmarking efforts, promote consistency, and provide a solid foundation for the broader adoption of LLMs in various applications.

ACKNOWLEDGEMENTS

Authors would like to thank Anna Joseph, Gogula Akhil Reddy , Arun Kumar Tiwary , Bhavana k, Divya Singh, Harshitha T, Vadla Sai Charitha, Sarthak Dwivedi, Kammara Prasad Achari, Arunima Divya, who are engineers from InfobellIT who helped test and develop this benchmark.

REFERENCES

[1] https://spec.org/
[2] https://tpc.org/
[3] Raghunath Nambiar, Tilmann Rabl, Karthik Kulkarni, Michael Frank:
Enhancing Data Generation in TPCx-HS with a Non-uniform Random Distribution. TPCTC: 2015: 94-129
[4] Meikel Poess, Raghunath Nambiar, Karthik Kulkarni, Chinmayi Narasimhadevara, Tilmann Rabl, Hans-Arno Jacobsen: Analysis of TPCx-IoT: The First Industry Standard Benchmark for IoT Gateway Systems. ICDE 2018: 1519-1530
[5] https://www.intel.com/content/www/us/en/developer/articles/technical/accelerate-llama2-ai-hardware-sw-optimizations.html
[6] https://huggingface.co/NousResearch/Llama-2-7b-hf?ref=blog.truefoundry.com
[7] https://github.com/huggingface/text-generation-inference
[8] https://locust.io/
[9] Llama 2: Open Foundation and Fine-Tuned Chat Models - https://arxiv.org/pdf/2307.09288.pdf
[10] https://www.anyscale.com/blog/reproducible-performance-metrics-for-llm-inference
[11] EchoSwift: https://github.com/Infobellit-Solutions-Pvt-Ltd/EchoSwift