The Importance of Workload Choice in Evaluating LLM Inference Systems

Konstantinos Papaioannou
IMDEA Software Institute
Universidad Politécnica de Madrid
konstantinos.papaioannou@imdea.org

Thaleia Dimitra Doudali
IMDEA Software Institute
Madrid, Spain
thaleia.doudali@imdea.org

Abstract

The success of Large Language Models (LLMs) across a wide range of applications and use cases has created the need for faster and more scalable systems for LLM inference. These systems speed up LLM inference by optimizing scheduling decisions or efficiently managing the available memory. However, most of them use synthetic datasets and target latency-critical scenarios in their evaluation, thereby overlooking a considerable part of real-world use cases and workloads. As a response, this paper presents an extensive experimental evaluation that aims to capture the impact of the workload used for evaluation and quantify the benefit derived from higher memory availability. Our analysis shows that LLMs can achieve 3× higher throughput for text generation and question-answering use cases compared to text summarization and conversational ones. The latter ones seem to exhibit low levels of performance due to their demanding input sizes. In addition, non-latency-critical inference services achieve 2.3× higher throughput when 4× more memory is available. In conclusion, this paper aims to highlight the importance and impact of the chosen workloads in the evaluation of systems for LLM inference.

CCS Concepts: • **Computing methodologies → Machine learning**.

Keywords: Large Language Models, Inference, Machine Learning, KV Cache

ACM Reference Format:
Konstantinos Papaioannou and Thaleia Dimitra Doudali. 2024. The Importance of Workload Choice in Evaluating LLM Inference Systems. In *4th Workshop on Machine Learning and Systems (EuroMLSys '24), April 22, 2024, Athens, Greece.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/xxxxxxx.xxxxxxx

EuroMLSys '24, April 22, 2024, Athens, Greece

ACM ISBN 978-x-xxxx-xxxx-x/23/01…$15.00
https://doi.org/10.1145/xxxxxxx.xxxxxxx

1 Introduction

Large Language Models (LLMs) are becoming more and more popular, powering a wide range of use cases and applications, including text generation [21] and summarization [11], question-answering [12] and conversation-based [41] personal assistants. Fast and scalable LLM inference is needed to manage millions of user requests at the same time. While the use of custom hardware, such as inference accelerators, is beneficial, it is still not sufficient to meet the required performance levels. Hence, custom inference engines are becoming increasingly essential.

LLMs present a unique opportunity for accelerating their inference, due to the input processing happening in two distinct phases and the model's internal data structures having a particular organization in memory. Thus, recently proposed custom LLM inference engines [1, 17, 24, 25, 31, 38, 52] are able to increase performance by optimizing scheduling decisions during input processing and by managing the available memory more efficiently.

We extensively study these papers, with a particular focus on the workloads they use for evaluation. We observe that most of them evaluate performance using synthetic datasets, that try to emulate use cases like text summarization, while targeting latency-critical performance requirements. Unfortunately, this approach significantly limits the evaluation to a restricted subset of real-world use cases and workloads for LLM inference systems.

Motivated by this observation, this paper conducts an extensive performance evaluation across different LLM types and sizes, using various public datasets that capture a range of real-world use cases. In addition, we quantify the benefit of higher memory availability across models and datasets. Our analysis reveals that the dataset and, therefore, the use case of the LLM directly impact the attainable levels of performance, due to the distinct characteristics of the input and output data. For instance, we notice that current LLM inference systems deliver 3× higher throughput for text generation and question-answering use cases compared to text summarization and conversational ones. In the latter cases, current works can only serve a very limited number of requests per second, highlighting the need for potentially custom future approaches. Finally, our analysis shows that a 4× increase in memory availability leads to a 2.3× higher throughput for workloads without strict latency requirements.

The remainder of the document is organized as follows. Section 2 provides background information on LLM inference and presents popular applications and use cases. Section 3 presents the different datasets and models that we will use in our experimental analysis. Section 4 analyzes the impact of use cases and memory availability on the performance of LLM inference systems. Section 5 summarizes important lessons learned from the analysis and discusses how they relate to current works.

2 Background

This section introduces basic terminology regarding Large Language Models (LLMs), presents their model architecture, explains the inference process, and describes applications and use cases powered by LLMs.

LLM Architecture and Inference. LLMs are deep neural networks following the Transformer architecture, eliminating the need for convolutions or recurrence as in other model types. Their success lies in the attention mechanism [49], that allows the model to focus on specific sections of the input and capture relations and dependencies. A typical LLM architecture consists of a stack of identical layers, where each layer is primarily comprised of a self-attention and a feed-forward layer. Finally, regarding their overall size, popular LLMs, such as GPT [3], OPT [55], and Llama-2 [47], consist of a few tens up to hundreds of layers, and a few hundred million up to a trillion of parameters.

LLM inference consists of two phases. Initially, the model transforms the input data into a sequence of small pieces called tokens, initiating the **prompt phase**. Then, the input tokens $(x_1, x_2, ..., x_n)$ are processed collectively, generating the Key-Value (KV) cache for each layer of the LLM. In more detail, the input tokens, represented as fixed-size vectors, form a matrix that is multiplied with three pretrained weight matrices, W_Q, W_K, W_V, generating the Query (Q), Key (K), and Value (V) matrices. Next, the model uses the Q, K, V matrices to compute the final output of the self-attention layer. This output then goes through the feed-forward layer, and the resulting output becomes the input for the next LLM layer. This process continues until the initial input traverses through the last layer. At the end of the prompt phase, the model has generated the KV cache, which is the collection of K and V matrices of every layer. This phase is known to be compute-bound, involving multiple matrix-matrix multiplications, and it can efficiently leverage GPU parallelism [25, 31].

Next is the **generation phase**, which utilizes the previously created KV cache to generate new tokens, one at a time. The generation of new tokens depends on all the previous ones, including both the input tokens and the previously generated tokens. Specifically, it relies on the KV cache produced by these tokens. Each time a new token is generated, new key and value vectors are appended to the KV cache increasing its size. The generation phase terminates when an end-of-sequence token is generated or when the maximum sequence length is reached. This phase is known to be memory-bound due to the absence of computationally intensive operations and the gradually increasing size of the KV cache [1, 24, 25, 31].

LLM-powered Applications. LLMs are often available to end-users through inference services. User-facing applications or users can directly interact with these services, submitting requests and receiving responses. Based on the type of application and user interaction, the LLM may need to handle different workloads and satisfy various performance objectives. On one hand, interactive applications, like email assistants or chatbots, have tight latency constraints and experience varying levels of activity throughout the day, thus generating different volumes of incoming requests to the underlying LLM. In the rest of this paper, we refer to these applications as **latency-critical**. On the other hand, "back-of-house" tasks, such as report summarization or sentiment analysis of reviews, lack strict latency requirements and can be executed in batches to increase throughput [38]. In the rest of this paper, we refer to these applications as **best-effort**.

LLM Use Cases. The applications mentioned above, support a broad spectrum of use cases that determine the size of the input to the LLM and the generated output by the LLM. One popular use case is **text generation**, including applications that write articles, blogs and social media content [18, 21, 27], or even generate stories and poems [10, 22, 23]. These applications typically take short sequences as input and produce larger outputs. In contrast, in **text summarization**, applications that summarize articles, business documents or research publications [11, 14, 34] receive large inputs and produce shorter outputs. Another common use case is **question-answering**, as seen in numerous LLM-based customer support chatbots [20, 53] and health-related virtual assistants [5, 12]. In these cases, the input sequences are usually short questions, and the produced output varies based on the question type. Similarly, in **conversational** use cases, which are identified in general-purpose platforms such as OpenAI [30] or ShareGPT [41], users interact with the platform in a variety of ways, resulting in input and output sequences of arbitrary length. Finally, other notable LLM use cases are code generation [19, 29, 36], sentiment analysis [2, 37] and natural language translation [8, 13, 28].

3 Characterization

This section presents the different datasets and models that we will use in our experimental analysis. It describes characteristics of the selected datasets and models, with a focus on the use cases and corresponding sizes.

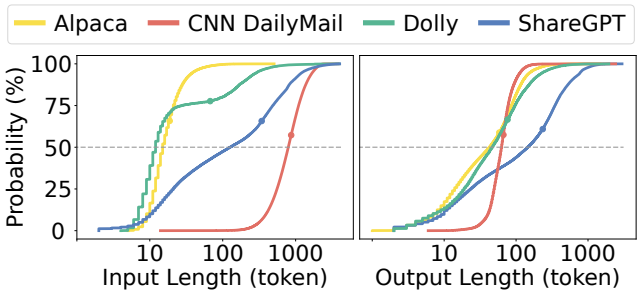


Figure 1. Cumulative distribution function (CDF) of input and output sequence lengths across datasets.

3.1 Datasets

In our analysis, we select four different datasets (Alpaca, CNN-DailyMail, Dolly, ShareGPT) which cover popular use cases (see Table 1 and Section 2). First, Alpaca [39] is a text generation dataset of 52k instructions and their corresponding responses. Second, CNN DailyMail [16, 35] is a text summarization dataset with 300k articles and their highlights. Third, Dolly [7] is a dataset of 15k instruction-following records classified in behavioral categories. We create a question-answering dataset of 9k records using the closed QA, open QA, and general QA categories. The closed QA category also includes the context of the question which we exclude and consider only the question. Finally, ShareGPT [41] is a dataset collected from the ShareGPT platform and has multi-round conversations between humans and ChatGPT. We consider this a conversational dataset by extracting the user's first input and ChatGPT's first response.

As mentioned in Section 2, the input and output sequences may exhibit very different lengths based on the use case. Figure 1 captures this difference for the four chosen datasets. It shows the cumulative distribution function (CDF) of the input (left figure) and output (right figure) sequence lengths across datasets. The x-axis of the figures captures the sequence lengths in terms of the number of tokens, and it is in logarithmic scale to better portray the wide range of lengths.

We observe that the Alpaca and Dolly datasets exhibit shorter input sequences compared to the corresponding output sequences. This is because the Alpaca dataset, in particular, contains as input a variety of short task instructions, such as "write a story," "generate a shopping list," etc., the majority of which generate longer sequences as output [39]. Similarly, within the different QA classes of the Dolly dataset, most questions are short and produce long answers [7].

Conversely, the CNN DailyMail dataset has the largest input sequences among all datasets, nearly three orders of magnitude larger when considering the logarithmic scale of the x-axis. The outputs have similar size to the rest of the datasets and are almost two orders of magnitude smaller than the input. This happens because the dataset corresponds to a text summarization use case, where articles are larger than their condensed highlights. Lastly, the ShareGPT dataset exhibits a smooth slope for both input and output, indicating a more uniform distribution of sequence lengths. As this is a dataset of conversations, the lengths of both input and output vary based on the conversation's topic.

Table 1. Dataset classification

Dataset	Use Case
Alpaca	text generation
Dolly	question-answering
CNN DailyMail	text summarization
ShareGPT	conversational

Takeaways. Our analysis shows that the LLM use case directly impacts the properties of both the input data and the generated output, resulting in significant differences in their sequence length distributions. At times, these differences span **orders of magnitude**, as seen between the text generation and text summarization use cases. Furthermore, we observe that input sequences are more sensitive to the specific use case, compared to the output ones, due to the unique characteristics of each use case. The insights from the above analysis will later help us explain the variations observed in the performance of LLM inference services.

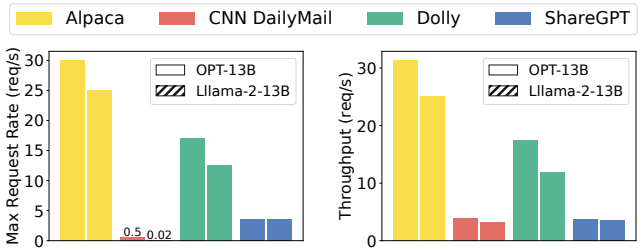
3.2 Models

From the plethora of LLMs, we experiment with the open-source and widely used models, OPT [55] and Llama-2 [48]. We choose two size variations of 13B and nearly 7B parameters for each one. Particularly, we select the pretrained versions of OPT-6.7B [44], OPT-13B [43], Llama-2-7B [46] and Llama-2-13B [45] from the Hugging Face model repository.

First, models with 13B parameters are commonly used to evaluate recently proposed systems designed to improve LLM inference [1, 17, 24, 25, 31, 38, 52]. We will delve deeper into these systems in Section 5.2. In addition, models with 13B parameters generate a memory footprint that nearly reaches the maximum memory capacity of an NVIDIA A100 40GB GPU. We use this type of GPU in our experiments (see Section 4.1) as it is also commonly used across related works.

Second, the 7B parameter class has over 50% fewer parameters and needs around 50% less memory, allowing experimentation with models half the size. Smaller models are cheaper and faster to train and serve [32, 48], mitigating the problem of GPU scarcity [15]. In addition, they excel in specific use cases, such as sentiment analysis [26], and are sometimes preferred over larger models when considering various combinations of latency and accuracy goals [33, 54]. Most importantly for us, we want to see whether we will observe common behaviors across two distinct model sizes, where one is 2× larger than the other.

Finally, it is notable that these models are versatile and can be used across multiple use cases and applications, like



(a) Latency-critical Inference **(b)** Best-effort Inference

Figure 2. Performance of the vLLM inference engine for different inference scenarios.

the ones covered by the datasets of Section 3.1, as seen in the HELM Classic Leaderboard [9]. Additionally, these models can be fine-tuned for specific use cases, meaning that they are retrained on datasets tailored to the respective use case, thereby enhancing their accuracy in that specific context.

4 Experimental Analysis

This section describes our experimental setup (Section 4.1), followed by an extensive experimental analysis exploring the impact of the use case (Section 4.2) and size of the KV cache (Section 4.3) on the performance of LLM inference.

4.1 Experimental Setup

We experiment on a server with native hardware that includes one NVIDIA A100 GPU with 40GB memory. We deploy the recently proposed vLLM inference engine [51], designed particularly for LLMs and aiming to improve throughput without sacrificing latency. In particular, vLLM leverages PagedAttention [25], a novel attention algorithm inspired by classic operating system techniques, such as virtual memory and paging. PagedAttention organizes the KV cache into fixed-size blocks, allowing for non-contiguous and dynamic memory allocations, in contrast to previous approaches that preallocate all the necessary memory for the KV cache. One block can have multiple key-value vectors, based on its size.

To control the GPU memory used for the KV cache we configure the maximum number of blocks to be allocated, as done for the experiments of Section 4.3. We configure the block size to be 16, as it was shown to provide the best trade-off between block size and performance [25]. Finally, vLLM provides modified versions of the OPT [55] and LLaMA [47] models, that include the PagedAttention mechanism. We open-source on GitHub[1] an extended version of vLLM, which we used for the experimental analysis, together with the source code for the experiments.

Performance Metrics. We conduct experiments targeting the two distinct types of LLM-powered applications, latency-critical and best-effort inference, as described in Section 2.

[1] https://github.com/muse-research-lab/llm-inference-workload-eval

The target performance metric in latency-critical inference is the maximum request rate (requests/sec) that the inference engine can handle under a specific service-level objective (SLO). In our case, we pick a p95 (95th percentile) latency SLO, meaning that the latency should be below a specified threshold for 95% of the requests. Given that an average English reader can read 4 words per second [4] and 1 token equals to 0.75 words [40], we set the SLO as 0.1875 seconds per generated token as also used in the S^3 paper [24]. We report the maximum request rate achieved without violating the SLO, after experimenting with increasing request rates. The request arrival times follow a Poisson distribution, as it has been shown to resemble real-world behaviors [6, 56].

The target performance metric in best-effort inference is the throughput (requests/sec) that the inference engine can sustain when there are no strict latency requirements and SLOs. For this reason we set as input a batch with a fixed number of requests, 1000 in our case, and we capture the time it takes to process all of them.

4.2 Impact of Use Case on Performance

This experiment aims to capture how various LLM use cases, with distinct input and output sequence length distributions, impact LLM inference performance. Thus, we create Figure 2, which presents performance for two distinct scenarios. On the left, Figure 2a pertains to the latency-critical scenario and presents, on the y-axis, the maximum request rate that does not violate the SLO. On the right, Figure 2b refers to the best-effort scenario displaying the throughput for 1000 requests. In both figures we evaluate performance combining OPT-13B and Llama-2-13B models with the four datasets. The bars of Llama-2-13B are distinguished by the hatches.

Regarding the latency-critical scenario, we observe a significant performance difference based on the dataset, namely the LLM use case. In more detail, datasets with shorter input sequences, such as Alpaca and Dolly, achieve over a 3.5× higher request rate compared to datasets with longer inputs, such as CNN DailyMail and ShareGPT (see Figure 1). This happens because the prompt phase is compute-bound; long input sequences can significantly prolong the duration of this phase, thus impacting the total response time. Particularly noteworthy is the case of CNN DailyMail, where we observe very low maximum request rates for both OPT-13B and Llama-2-13B. As shown in Figure 1, the input sequences in this dataset are larger by three orders of magnitude compared to the other datasets. Consequently, the prompt phase and the average time to serve one token get severely prolonged, causing SLO violation. Similar behaviors have been observed and explained extensively in related literature [17, 50].

Regarding the best-effort scenario, we observe similar trends due to the difference in the input length distributions. In particular, Alpaca and Dolly, achieve over 3× higher throughput compared to CNN DailyMail and ShareGPT, regardless of the model. In the case of CNN DailyMail, although

the throughput appears relatively low, in reality we calculate that 3.5 thousand tokens per second are still being processed, as one input sequence may have thousands of tokens.

In both scenarios, distinguishing the impact of the output length is challenging because it exhibits a similar distribution across all datasets (see Figure 1). Furthermore, we observe that the type of model can result in different LLM inference performance. For instance, OPT-13B outperforms Llama-2-13B in both scenarios by an average of 1.2×. The difference is more evident for the Alpaca and Dolly datasets and almost negligible for the CNN DailyMail and ShareGPT datasets.

Takeaways. The above analysis highlights the direct impact of the use case on the performance of LLM inference. Overall, we see that LLMs can achieve 3× higher throughput for text generation and question-answering use cases compared to text summarization and conversational ones, in both latency-critical and best-effort scenarios. In particular, text summarization sustains very low throughput due to the significantly larger input sequences. Finally, the type of LLM model used introduces a performance difference of 1.2× for text generation and question-answering use cases, while it shows negligible impact on text summarization and conversational use cases.

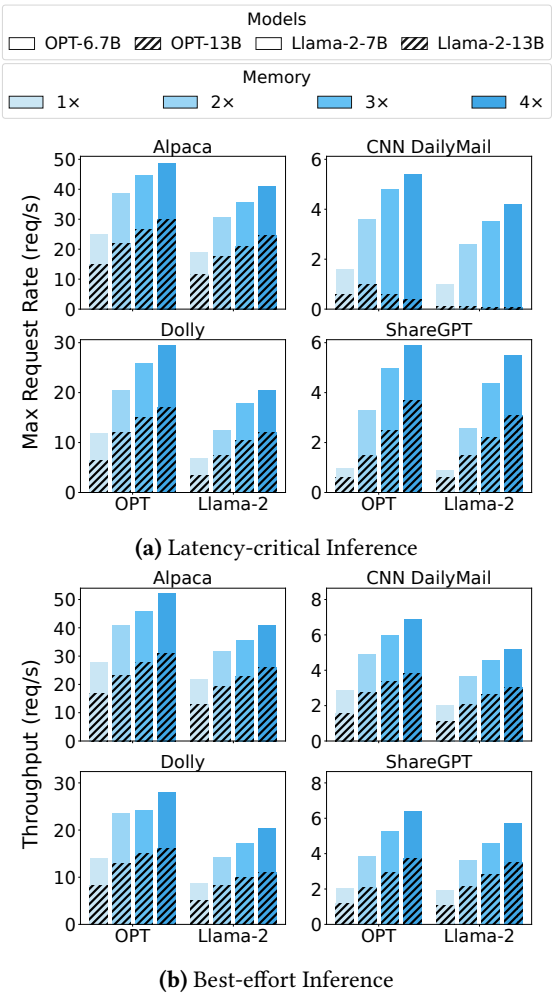
4.3 Impact of KV Cache Size on Performance

The following experiments aim to analyze and quantify the impact of the KV cache size and memory availability on LLM inference performance. Figures 3a and 3b show performance under increasing amounts of available KV cache memory (1× - 4×), across datasets and models, for the latency-critical and best-effort scenarios, respectively. The bars of the 13B models are distinguished by the dotted hatches.

The experiment starts with a KV cache size of 150 KV blocks and increases in increments of 150 blocks, reaching a maximum of 600 KV blocks. We set 600 blocks as the upper limit since that configuration results in OPT-13B and Llama-2-13B occupying 90-95% of the available 40 GB GPU memory. The initial memory footprint is 1.83 GB of KV cache for OPT-13B and Llama-2-13B, and 1.17 GB for OPT-6.7B and Llama-2-7B, due to the smaller size.

Latency-critical Inference. In Figure 3a, we observe that across different models and datasets, increasing the KV cache size results in higher performance, in most cases. The benefit is greater for 1× to 2× more memory and then diminishes from 2× to 4× more. In the case of both Alpaca and Dolly, having 4× more KV cache size enables over a 2× increase of the maximum request rate. The benefits of higher memory availability apply to both model types and sizes.

Similarly, ShareGPT shows increasing performance with an increased KV cache size, reaching a notably higher request rate, 5-6× higher, for a 4× larger KV cache. However, while the relative increase is significant, the actual increase remains low. For instance, the request rate of Llama-2-7B increases



(a) Latency-critical Inference

(b) Best-effort Inference

Figure 3. Maximum request rate of latency-critical inference services under a given SLO, and throughput of best-effort inference services, considering varying sizes of KV cache, different models, and datasets.

from 0.9 req/s to 5.5 req/s. Thus, more available memory doesn't necessarily translate to high performance, if the use case itself cannot sustain it, as discussed in Section 4.2.

Lastly, the CNN DailyMail dataset shows again a unique behavior. Surprisingly, in the case of both 13B models, we observe *negligible* differences in performance regardless of memory availability. As explained in Section 4.2, longer input sequences lead to longer prompt phases, resulting in increased request latency. However, smaller models, such as OPT-6.7B and Llama-2-7B, execute more quickly, and their prompt phases are less computationally demanding than those of larger models. Thus, we observe a performance increase proportional to the increase in the KV cache size. For instance, the request rate increases by an average of 3.8× across the 7B models when the KV cache is 4× larger.

Best-effort Inference. In Figure 3b, we see that across all combinations of models and datasets, a 4× increase of the KV cache results in a 2.3× higher throughput, on average. In particular, the performance boost is more significant in datasets with large input or output sequence lengths. For instance, CNN DailyMail and ShareGPT datasets show, on average, a 2.6× higher throughput compared to the 2× observed for the Alpaca and Dolly datasets, with a 4× larger KV cache. In addition, we see similar relative throughput increases regardless of the model type and size. This is attributed to best-effort inference attempting to serve as many requests possible simultaneously. Thus, the more memory available the more requests can be served.

Takeaways. Overall, we see that higher memory availability almost always improves performance. This improvement is on average 3.2× for latency-critical scenarios and 2.3× for best-effort, when having 4× more memory. However, there are use cases such as the text summarization and conversational that in general sustain low performance, which doesn't change drastically with more memory availability, especially in latency-critical scenarios.

5 Discussion

5.1 Lessons Learned

Overall, our analysis shows that the amount of available memory and the use case of the LLM, especially the properties of the input and output data, significantly impact the attainable levels of inference performance. We conclude that:

- Text generation and question-answering are use cases that achieve high performance across both latency-critical and best-effort scenarios. In particular, they significantly benefit from larger memory availability for their KV cache. Therefore, these use cases are ideal for evaluating systems solutions that improve the LLM memory management.
- Best-effort inference consistently benefits from higher memory availability. Therefore, there is great value in focusing on enhancing memory management, particularly for this inference scenario.
- Text summarization and conversational use cases are extremely demanding, due to properties of their input data. They exhibit consistently low performance irrespective of the model type and size, the inference scenario and size of the KV cache. Therefore, it is essential to treat these use cases separately and potentially apply completely different approaches to improve their inference performance.

5.2 Systems for LLMs

This section summarizes recently proposed systems that improve upon LLM inference, focusing on their approach and workloads used for evaluation.

One approach that related works use to accelerate LLM inference is to optimize the scheduling of the prompt vs. the generation phase. In more detail, Orca [52] was the first to introduce iteration-level scheduling and fuse the execution of new requests alongside existing ones. Taking a step further, SARATHI [1] and DeepSpeed-FastGen [17] study the distribution of input and output sequence lengths to find an optimal interleaving of the prompt and generation phases across requests. In contrast, Splitwise [31] chooses to execute the two phases on completely separate machines due to their distinct characteristics.

An additional approach for improving LLM inference is to efficiently manage memory, primarily the model's KV cache. For instance, vLLM [51] augments the iteration-level scheduling proposed by Orca [52] with block-level memory management via PagedAttention [25], as detailed in Section 4.1. Another example is S^3 [24], which preallocates only the necessary memory for the KV cache, by predicting the output sequence length. Finally, FlexGen [38] improves high-throughput LLM inference under limited resources by designing offloading strategies for the KV cache, considering memory availability across GPU, CPU and disk.

Regarding the evaluation of the above systems, we first notice that most of them use synthetic datasets, and secondly, that they focus on latency-critical inference scenarios. The synthetic datasets mostly consist of longer inputs and shorter outputs, resembling some properties of the text summarization use case, as done in DeepSpeed-FastGen [17] and FlexGen [38] for latency-critical and best-effort scenarios, respectively. Orca [52] performs a comprehensive evaluation with a broad range of data characteristics and inference scenarios, while SARATHI [1] evaluates against Orca. In contrast, vLLM [51] evaluates against real-world latency-critical conversational and text generation use cases using the public datasets Alpaca [39] and ShareGPT [41]. Similarly, Splitwise [31] focuses on real-word latency-critical coding and conversational use cases with private production traces from Microsoft Azure. Finally, S^3 [24] evaluates both latency-critical and best-effort inference scenarios targeting the text generation use case captured in the Alpaca [39] dataset.

Overall, we observe that most related works do not reason about their evaluation workloads. They primarily evaluate against synthetic datasets and latency-critical workloads. Such evaluations may capture only some of the real-world behaviors of LLM inference systems. Instead, our analysis shows that different real-world use cases can reach different levels of performance and are sensitive to the memory availability for the model's KV cache. In conclusion, this paper aims to highlight the importance of extensively evaluating systems for LLMs against a wide range of real use cases, using the abundance of public datasets available [42].

6 Summary

This paper presents extensive experimental analysis capturing the impact of workload and available memory on the

evaluation of systems built to accelerate LLM inference services. Our analysis reveals that text generation and question-answering use cases achieve higher throughput compared to text summarization and conversational ones. For the latter ones, current systems fail to achieve substantial performance levels due to their vastly larger input size. Finally, our analysis highlights significant benefit from higher memory availability, particularly in inference without strict latency requirements.

Acknowledgments

We thank the reviewers for their constructive feedback. This work is part of the grants FJC2021-047102-I, TED2021-132464B-I00, PID2022-142290OB-I00, funded by the European Union «NextGenerationEU»/PRTR, the ESF+ and MCIN/AEI/10.13039/501100011033.

References

[1] Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, and Ramachandran Ramjee. 2023. SARATHI: Efficient LLM Inference by Piggybacking Decodes with Chunked Prefills. arXiv:2308.16369 [cs.LG]

[2] OnSearch Pty Ltd T/A Relevance AI. 2023. Relevance AI. https://relevanceai.com.

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models Are Few-Shot Learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS'20)*. Curran Associates Inc., Red Hook, NY, USA, Article 159, 25 pages.

[4] Marc Brysbaert. 2019. How many words do we read per minute? A review and meta-analysis of reading rate. *Journal of Memory and Language* 109 (2019), 104047. https://doi.org/10.1016/j.jml.2019.104047

[5] PACT Care BV. 2019. Florence. https://florence.chat.

[6] Seungbeom Choi, Sunho Lee, Yeonjae Kim, Jongse Park, Youngjin Kwon, and Jaehyuk Huh. 2022. Serving Heterogeneous Machine Learning Models on Multi-GPU Servers with Spatio-Temporal Sharing. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. USENIX Association, Carlsbad, CA, 199–216. https://www.usenix.org/conference/atc22/presentation/choi-seungbeom

[7] Mike Conover, Matt Hayes, Ankit Mathur, Jianwei Xie, Jun Wan, Sam Shah, Ali Ghodsi, Patrick Wendell, Matei Zaharia, and Reynold Xin. 2023. *Free Dolly: Introducing the World's First Truly Open Instruction-Tuned LLM.* https://www.databricks.com/blog/2023/04/12/dolly-first-open-commercially-viable-instruction-tuned-llm

[8] DeepL. 2024. DeepL Translator. https://www.deepl.com/translator.

[9] Stanford Center for Research on Foundation Models. 2023. HELM Classic Leaderboard. https://crfm.stanford.edu/helm/classic/latest/#/leaderboard.

[10] AI Poem Generator. 2024. AI Poem Generator. https://www.aipoemgenerator.org.

[11] GetDigest. 2024. GetDigest. https://getdigest.com/en.

[12] Ada Health GmbH. 2024. Ada. https://ada.com.

[13] Google. 2024. Google Translate. https://translate.google.com/.

[14] Grammarly. 2024. AI Summarizing Tool. https://www.grammarly.com/summarizing-tool.

[15] Erin Griffith. 2023. The Desperate Hunt for the A.I. Boom's Most Indispensable Prize. https://www.nytimes.com/2023/08/16/technology/ai-gpu-chips-shortage.html.

[16] Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1* (Montreal, Canada) *(NIPS'15)*. MIT Press, Cambridge, MA, USA, 1693–1701.

[17] Connor Holmes, Masahiro Tanaka, Michael Wyatt, Ammar Ahmad Awan, Jeff Rasley, Samyam Rajbhandari, Reza Yazdani Aminabadi, Heyang Qin, Arash Bakhtiari, Lev Kurilenko, and Yuxiong He. 2024. DeepSpeed-FastGen: High-throughput Text Generation for LLMs via MII and DeepSpeed-Inference. arXiv:2401.08671 [cs.PF]

[18] Frase Inc. 2024. Frase. https://www.frase.io/.

[19] GitHub Inc. 2024. GitHub Copilot. https://github.com/features/copilot.

[20] Hubspot Inc. 2024. HubSpot. https://www.hubspot.com.

[21] Pepper Content Inc. 2024. Peppertype.ai. https://www.peppercontent.io/peppertype-ai/.

[22] Writesonic Inc. 2024. WriteSonic. https://writesonic.com/.

[23] Jasper.ai. 2024. Jasepr AI. https://www.jasper.ai/.

[24] Yunho Jin, Chun-Feng Wu, David Brooks, and Gu-Yeon Wei. 2023. S^3: Increasing GPU Utilization during Generative Inference for Higher Throughput. arXiv:2306.06000 [cs.AR]

[25] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles* (Koblenz, Germany) *(SOSP '23)*. Association for Computing Machinery, New York, NY, USA, 611–626. https://doi.org/10.1145/3600006.3613165

[26] Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Alexander Cosgrove, Christopher D Manning, Christopher Re, Diana Acosta-Navas, Drew Arad Hudson, Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue WANG, Keshav Santhanam, Laurel Orr, Lucia Zheng, Mert Yuksekgonul, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri S. Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Andrew Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. 2023. Holistic Evaluation of Language Models. https://openreview.net/forum?id=iO4LZibEqW Featured Certification, Expert Certification.

[27] Rytr LLC. 2024. Rytr - Best AI Writer. https://rytr.me/.

[28] Microsoft. 2024. Search Microsoft Translator. https://www.bing.com/translator.

[29] Microsoft. 2024. Visual Studio Intelli Code. https://visualstudio.microsoft.com/services/intellicode/.

[30] OpenAI. 2023. ChatGPT. https://chat.openai.com.

[31] Pratyush Patel, Esha Choukse, Chaojie Zhang, Íñigo Goiri, Aashaka Shah, Saeed Maleki, and Ricardo Bianchini. 2023. Splitwise: Efficient generative LLM inference using phase splitting. arXiv:2311.18677 [cs.AR]

[32] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2022. Efficiently Scaling Transformer Inference. arXiv:2211.05102 [cs.LG]

[33] Francisco Romero, Qian Li, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2021. INFaaS: Automated Model-less Inference Serving. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 397–411. https://www.usenix.org/conference/atc21/presentation/romero

[34] Scribbr. 2024. Text Summarizer. https://www.scribbr.com/text-summarizer/.

[35] Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get To The Point: Summarization with Pointer-Generator Networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Vancouver, Canada, 1073–1083. https://doi.org/10.18653/v1/P17-1099

[36] Amazon Web Services. 2024. Amazon CodeWhisperer. https://aws.amazon.com/codewhisperer/.

[37] Amazon Web Services. 2024. Amazon Comprehend. https://aws.amazon.com/comprehend/.

[38] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU. In *Proceedings of the 40th International Conference on Machine Learning* (Honolulu, Hawaii, USA) *(ICML'23)*. JMLR.org, Article 1288, 23 pages.

[39] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford Alpaca: An Instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca.

[40] OpenAI Team. 2024. What are tokens and how to count them? https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-them. Accessed: 2024-02-01.

[41] ShareGPT Team. 2023. ShareGPT. https://sharegpt.com/.

[42] The Hugging Face team. 2016. Hugging Face Datasets. https://huggingface.co/datasets.

[43] The Hugging Face team. 2022. OPT-13B. https://huggingface.co/facebook/opt-13b.

[44] The Hugging Face team. 2022. OPT-6.7B. https://huggingface.co/facebook/opt-6.7b.

[45] The Hugging Face team. 2023. Llama-2-13B. https://huggingface.co/meta-llama/Llama-2-13b-hf.

[46] The Hugging Face team. 2023. Llama-2-7B. https://huggingface.co/meta-llama/Llama-2-7b-hf.

[47] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL]

[48] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL]

[49] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[50] vLLM Team. 2023. Notes on vLLM v.s. DeepSpeed-FastGen. https://blog.vllm.ai/2023/11/14/notes-vllm-vs-deepspeed.html.

[51] vLLM Team. 2024. vLLM: Easy, Fast, and Cheap LLM Serving with PagedAttention. https://vllm.ai. Accessed: 2024-02-01.

[52] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A Distributed Serving System for Transformer-Based Generative Models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 521–538. https://www.usenix.org/conference/osdi22/presentation/yu

[53] Zendesk. 2024. Zendesk. https://www.zendesk.es/.

[54] Jeff Zhang, Sameh Elnikety, Shuayb Zarar, Atul Gupta, and Siddharth Garg. 2020. Model-Switching: Dealing with Fluctuating Workloads in Machine-Learning-as-a-Service Systems. In *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*. USENIX Association. https://www.usenix.org/conference/hotcloud20/presentation/zhang

[55] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. OPT: Open Pre-trained Transformer Language Models. arXiv:2205.01068 [cs.CL]

[56] Yunqi Zhang, David Meisner, Jason Mars, and Lingjia Tang. 2016. Treadmill: attributing the source of tail latency through precise load testing and statistical inference. In *Proceedings of the 43rd International Symposium on Computer Architecture* (Seoul, Republic of Korea) *(ISCA '16)*. IEEE Press, 456–468. https://doi.org/10.1109/ISCA.2016.47