INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2025, VOL. 71, NO. 3, PP. 1–8
Manuscript received February 5, 2025; revised June, 2025. doi: 110.24425/ijet.2025.153620

Infrastructure for the deployment of Large Language Models: challenges and solutions

Tomasz Walkowiak, and Bartosz Walkowiak

***Abstract*—Large Language Models are increasingly prevalent, and their capabilities are advancing rapidly due to extensive research in this field. A growing number of models are being developed, with sizes significantly surpassing 70 billion parameters. As a result, the ability to perform efficient and scalable inferences on these models is becoming crucial to maximize the utilization of valuable resources such as GPUs and CPUs. This thesis outlines a process for selecting the most effective tools for efficient inference, supported by the results of experiments. Additionally, it provides a comprehensive description of an end-to-end system for the inference process, encompassing all components from model inference and communication to user management and a user-friendly web interface. Furthermore, we detail the development of an LLM chatbot that leverages the function-calling capabilities of LLMs and integrates various external tools, including weather prediction, Wikipedia information, symbolic math, and image generation.**

***Keywords*—large language models; model deployment; continuous batching; chatbot; function-calling LLM**

I. Introduction

THE applications of large language models (LLM) are extensive, covering a wide range of text analysis and generation systems. Their capability to process natural language makes them valuable components in numerous software, highlighting the need for accessible programming APIs. A traditional approach, pioneered by OpenAI, involves providing access to various models through paid public interfaces.

However, relying on external APIs carries the risk of dependence on third-party providers, and processing data outside one's organization may be subject to legal restrictions and the potential for information leakage. Additionally, newly developed models available through platforms like Hugging Face are often not easily accessible via API for high-traffic demands. Consequently, it is worth considering the option of allowing access to models from within an organization's own infrastructure or through external solutions, particularly for institutions equipped with graphics cards.

This article explores two key issues: the computational efficiency of large language models during inference and the establishment of infrastructure for processing text documents with these models.

The paper is structured as follows: Section II reviews the technologies and research relevant to the topics discussed in this article. Section III describes the inference speed-up techniques employed and presents a series of experiments carried out to identify the most effective method for deploying large language models. Section IV offers an in-depth examination of the architecture for deploying LLMs. Finally, Section V presents the architecture of a Multi-Service Chatbot that, in addition to its standard LLM functionalities, incorporates features such as access to weather forecasts and image generation. We also address the challenges encountered during the implementation of the chatbot and propose solutions to these issues.

T. Walkowiak and B. Walkowiak are with Faculty of Information and Communication Technology, Wroclaw University of Science and Technology, Wroclaw, Poland (e-mail: tomasz.walkowiak@pwr.edu.pl).

II. Related Works

Large Language Models have become a significant and important resource used by many users. Models in this group are steadily increasing in capability as well as size, a phenomenon that leads to an increase in the computing power needed to run them, and therefore a decrease in accessibility for the average user. Furthermore, a continuous increase in the size of published models can be observed, with the core size of models increasing from 7B to 70B and even 340B[1]. Moreover, the amount of vRAM needed to run them is also increasing due to the rising length of supported contexts. However, many solutions exist and are being developed whose developers are trying to contribute to optimizing and speeding up the inference process on LLMs.

Methods such as quantization reduce computational costs by reducing the precision of the operations performed. The use of this approach allows large language models to be run with limited resources, such as graphics card memory. The disadvantages of this solution are slower generation speeds and, in extreme cases, significant degradation in the quality of the model output. [1], [2]

Further solutions focus on efficiency of the use of available resources. An example is the llama-cpp [3] project, based on the GGML [4] format, which enables large models to run efficiently on the CPU. Another solution is ONNX [5], which optimizes the use of GPU or CPU, allowing the process of generating successive tokens to be accelerated.

There are also approaches that increase the speed of inference through the use of techniques that parallelize computation. Many strategies for increasing the speed of computation through parallelism are provided by the Colossal-AI [6]

[1] https://huggingface.co/nvidia/Nemotron-4-340B-Instruct

toolkit. Among the available strategies are those such as auto-parallelism [7] or sequence-parallelism [8].

The last group of solutions, which is the most significant for the end user, is those that influence how available GPU resources are used. Such a technique is, among others, the ZeRO (Zero Redundancy Optimizer) technology, which combines model weights on the CPU or NVMe and only passes such prepared weights to the GPU for inference. This technique, despite its revolutionary nature, is not being developed. Other techniques that are being dynamically developed and contribute to optimizing the memory consumption of the graphics card are FlashAttention [9] and PagedAttention [10].

Large Language Models are capable of handling a wide range of tasks. However, they lack access to real-time information and, in some cases, existing tools may outperform LLMs. This issue can be addressed by integrating LLM with external tools [11], [12] through function calling [13]. Driven by practical applications, function calls can become quite complex due to the variety and complexity of the APIs available [12].

III. Engine Selection

In order to select and then implement an effective and efficient system for handling large language models, a comparison of available state-of-the-art solutions was carried out.

A. Inference Time Assessment

An important element of the research was the choice of a method to measure the inference time, which was the main performance indicator for the evaluated solutions. Available metrics for measuring generation speed include: Time To First Token, which allows us to compare the responsiveness of the solutions, Time Per Output Token, which includes the average time it takes to generate one response token, or Throughput, which is an extension of the previous metric, as it also takes into account the speed-ups resulting from multi-query combination techniques.

However, the metrics mentioned do not provide a comprehensive evaluation of model inference techniques. As highlighted in [14], the average number of tokens generated can vary depending on the length of the generated sequence. Therefore, the performance of the inference system should be measured in terms of the average number of tokens per second across different sequence-length intervals. This approach ensures that the system's efficiency is consistent and not affected by anomalies within specific response length ranges, something that would be overlooked if only a single average across all lengths was considered.

In addition, the described approach for the measurement of metrics in intervals of different output lengths was applied to the metric of total processing time. The total query handling time consists of both the actual processing time and the queuing time. This approach is driven by the need to assess the actual performance of the methods from an end-user perspective; measuring processing times alone could be misleading and not give a complete picture of the technology under test.

B. Attention and Batching Techniques

Techniques to optimize computation of attention in transformer based large language models can focus on various bottlenecks that slow down the computation. One of the key factors for inference speed is the number of I/O operations to High Bandwidth Memory required to perform the processing. Reducing the frequency of these costly operations is what the FlashAttention [9] algorithm is focused on. The calculations are split into blocks, avoiding loading the entire matrix to calculate the attentions. For additional speed-up, the Softmax Normalization Factor, needed when calculating the attentions during the backward transition, has been moved to the highly efficient SRAM located on the GPU chip.

Another important factor is how the memory used during query processing is allocated. This memory can be divided into a static part, where the loaded model weights are located, and a dynamic part, the content of which changes during computation. The PagedAttention [10] technique concentrates on optimizing the use of the dynamic part of this memory and uses techniques that are significantly different from the standard approach. In approaches other than PagedAttention, the dynamic part of the memory is pre-allocated, up to a size corresponding to the maximum length of the output. In addition, this memory is continuously indexed, which significantly hinders shared computation. However, in the PagedAttention algorithm, the aforementioned disadvantages are solved by dividing the KV cache into blocks of fixed size, which are not stored in continuous memory. This makes it possible to share part of the computation and also to use the available memory more efficiently.

A technique that can speed up the computation of large numbers of requests is batching, which is widely used for models with different types of input, both images and languages. Batching involves grouping together multiple requests and processing them at the same time. This solution is directly tailored for the use of GPUs, which are designed to process a single instruction for multiple data (SIMD). Among the known approaches to this issue is static batching, in which requests are grouped together and then waited until all have been processed. This solution can be inefficient because the length of the computation cannot be predicted, so some computations will be performed faster, and some resources will be unused while waiting for the other computations to be completed. An approach that solves the aforementioned problem is continuous batching [15], in which new requests are passed to the computation when the previous ones have finished, so that individual slots are better filled. The use of continuous batching increases the efficiency of query processing by maximizing the use of available GPU resources.

C. Examined Methods

In order to select an inference method, we conducted research on advanced methods that support mechanisms for parallelization and batching of computation, namely Hugging-Face Text Generation Inference[2] and vLLM [16]. Both of

[2] https://github.com/huggingface/text-generation-inference

these methods are open and dynamically developed projects. HuggingFace CasualLM, a part of Transformers library [17], which is a popular and often the first solution of choice, has been chosen as a reference for the selected solutions.

1) HuggingFace CasualLM: Through widespread and frequent use, the HuggingFace-created Transformer library is a standard solution when it comes to running language models. Part of this library is the module responsible for running causal language models, using left-handed attention to predict the next token in a sequence. Due to the aforementioned prevalence, the CausalLM module is the base solution when running inferences on LLMs, but it does not support a number of solutions to optimize and speed up the process of next token generation.

2) HuggingFace Text Generation Inference: Another solution from HuggingFace that provides support for methods that optimize the inference process such as FlashAttention and PagedAttention is Text Generation Inference (TGI). The solution also supports techniques such as continuous batching, quantization, and streaming responses using Server Site Events. It is a collection of tools tailored to run Large Language Models simply and efficiently.

3) vLLM: A technology that also supports the use of the PagedAttention mechanism for efficient key and value memory management is vLLM. This open source project is under dynamic development and its authors have implemented support for a number of useful features. These solutions include continuous batching, which ensures that there are no significant slowdowns when processing large numbers of requests. The vLLM project also supports a number of model quantization techniques, including GPTQ [18] quantisations.

D. Experiments and Results

Experiments were conducted to compare the proposed methods in terms of inference speed and solution scalability. The tests were performed using an NVIDIA 80GB A100 graphics card. Given the variety of large language models with different parameter counts, we evaluated models ranging from the small TinyLlama [19] with one million parameters, to OpenChat [20] with the popular 7B parameter size, and up to the slightly larger Vicuna-13B [21].

During the speed and scalability experiments, a dataset consisting of short answer questions and selected questions from the TruthfulQA [22] collection was used. By structuring the set of questions in this way, it was possible to obtain responses of varying lengths, from very short to long, which was essential to measure speed according to the methodology adopted in our study. The collection contains 50 questions that were run many times during the experiments, in order to make the results not dependent on the randomness present in Large Language Models.

Inference speed was measured as the average speed of generating a response to a single prompt for response lengths falling within three ranges (up to 30 tokens, 31-300 tokens, above 300 tokens). Table I presents the results of the experiment comparing the inference speed across different methods. It can be seen that the vLLM library is the most efficient of the tested methods; in addition, the average speed of generation is constant within a given model, independent of the length of the response. In contrast, for the TGI solution, the average speed increases for longer responses, and the average speeds are even more than ten times lower compared to vLLM.

TABLE I
AVERAGE SPEED [TOKENS/S] AS A FUNCTION OF OUTPUT SIZE FOR THREE MODELS OF VARYING SIZES AND THREE DIFFERENT INFERENCE METHODS ANALYZED

Model	Output size in tokens	Method		
		HF	TGI	vLLM
TinyLlama (1B)	1-30	51	15	260
	31-300	51	60	290
	301-	52	95	280
OpenChat (7B)	1-30	38	4	83
	31-300	33	48	85
	301-	26	70	83
Vicuna (13B)	1-30	34	6	50
	31-300	30	34	50
	301-	15	44	49

The scalability of the tested solutions was measured by examining their resistance to a decrease in response speed under the influence of a large number of queries. To this end, the throughput of the models was measured under the influence of batches of different sizes. Query sets were created by randomly arranging a set of 2,500 prompts, which contained 50 prompts with different response lengths repeated 50 times. In each round of the experiment, responses to a set of queries were generated while maintaining a fixed batch size for each of the solutions tested. The result of the measurements, presented in Table II, is the average processing speed, calculated as the total number of tokens generated divided by the processing time of the entire set.

In terms of scalability, the weakest performance is shown by HF CausalLM, for which average speeds are 3 to even 30 times lower compared to the other solutions. This result of the experiment can be attributed to the influence of different types of batching, where the basic HF library uses static batching and the other methods use continuous batching. In addition, the memory consumption optimization techniques used within vLLM and TGI, such as PagedAttention, allow large batch sizes to be processed, where the HF CausalLM, which does not use the aforementioned techniques, ran out of graphics card memory.

Furthermore, the TGI method is significantly less efficient when processing multiple queries at the same time compared to vLLM, this relationship is particularly evident with larger batch sizes. Although both methods support the same attention optimization and batching techniques, the difference between them can be more than 2 times for the smaller models and up to more than 1.5 times for the largest model under investigation.

IV. INFRASTRUCTURE ARCHITECTURE DESIGN

A. Overview

Based on the experimental results presented in the previous section, we have designed and implemented the infrastructure

TABLE II
AVERAGE SPEED [TOKENS/S] AS A FUNCTION OF BATCH SIZE FOR THREE MODELS OF VARYING SIZES AND THREE DIFFERENT INFERENCE METHODS ANALYZED.

TinyLlama (1B)			
Batch size	HF	TGI	vLLM
1	52	117	283
2	98	216	527
16	189	1127	2683
32	149	1732	3940
64	127	2434	5145
128	106	2837	5358
256	OoM	2837	5565
512	OoM	2819	5550

OpenChat (7B)			
Batch size	HF	TGI	vLLM
1	30	82	85
2	36	155	162
16	29	762	1024
32	29	1348	1681
64	26	2110	2488
128	24	2344	3093
256	OoM	2364	3411
512	OoM	2005	3507

Vicuna (13B)			
Batch size	HF	TGI	vLLM
1	17	49	50
2	16	95	97
16	11	542	608
32	11	842	984
64	10	1171	1468
128	9	1167	1795
256	OoM	1164	1874
512	OoM	1167	1924

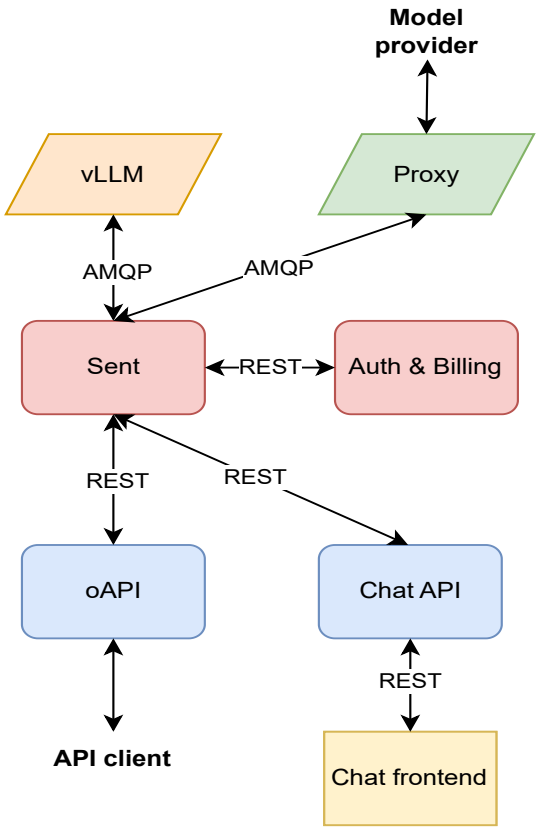

Fig. 1. Diagram of the architecture for LLM deployment with the type of communication between the elements.

for deploying Large Language Models using the vLLM library. However, several additional components, as illustrated in the schema in Fig. 1, have been incorporated. The system consists of:

1) *vLLM* modules for LLM inference,
2) *Proxy* modules for accessing external model providers (e.g., OpenAI),
3) *Sent* module functioning as a router that manages authorization,
4) *Auth & Billing* module for tracking model usage by users and enforcing access rules.

The system includes two APIs: the first (*oAPI*) adheres to the OpenAI specification[3], while the second, the *Chat API*, provides access to a web-based GUI, enabling conversational capabilities directly within a web browser.

B. Communication Protocols

Two communication techniques are used between the system components: REST and the AMQP[4] protocol. REST is the obvious choice, as it is the standard for any APIs. Due to the streaming nature of conversations with LLMs (where responses are generated token by token in real-time), Server-Sent Events (SSE), part of the HTML5 EventSource standard[5], are employed.

For communication between the Sent service and LLM services (*vLLM* and *Proxy*), we utilize a message broker, RabbitMQ[6], which implements the AMQP protocol. RabbitMQ is responsible for message queueing and enables simple horizontal scaling. Moreover, RabbitMQ offers a GUI interface that allows administrators to monitor request flow and load on LLM services over short time intervals (up to an hour), making system administration much more manageable.

C. Chat API and Frontend

Large Language Models are provided through a web GUI, which consists of a front-end application using the node JS framework and a back-end part exposing the API using the FastAPI [23] library in Python. The http requests triggered by user actions are handled by the backend, which is responsible for interaction with the inference engine on the models as well as user authentication and authorization.

[3] https://platform.openai.com/docs/api-reference/chat
[4] https://www.amqp.org
[5] https://html.spec.whatwg.org
[6] https://www.rabbitmq.com

As part of the architecture of the web GUI service, a non-relational MongoDB database is also used, in which the queries and answers of the models are stored, allowing access to the conversation history. A mechanism has also been implemented that allows for the export of each conversation to a spreadsheet, facilitating the analysis of the model's responses.

In addition, both the inference part on the models and the back-end and front-end applications of the Web window support the use of models in streaming mode, using the Server Site Events mechanism. The use of this technology ensures that the response of the models is displayed continuously and the waiting time for the start of the response is minimal.

The end user can take full advantage of the capabilities of the models provided, without worrying about implementation or setup, while still maintaining control over generation by adjusting parameters such as top_p, temperature and repetition penalty. As part of the interaction with the large language model, it is also possible to set the value of the system prompt.

D. System Usage

Development of the system began in April 2024, and within the next six months, it handled over 11 million prompts, generating more than 800 million tokens.

V. Multi-Service Chatbot

A. Introduction

Large Language Models are capable of solving a wide range of tasks. However, they lack access to real-time information (such as current weather) and, in some cases, existing tools offer superior capabilities (e.g. symbolic math packages). These limitations can be addressed by integrating LLMs with external tools through the use of function calling [13].

To illustrate the LLM and functional calling capabilities, we have developed a Multi-Service Chatbot[7] that, in addition to its standard LLM capabilities, is equipped with the following functionalities:

1) image generation,
2) weather forecasting,
3) solving mathematical equations,
4) accessing Wikipedia information.

During the design process, we assumed that only public models (i.e. models that could run in the infrastructure presented in Chapter IV could be utilized (thus excluding models such as GPT-4o). Additionally, we assumed that the chatbot would converse in Polish. To achieve this functionality, we require a dedicated reasoning module that analyzes user prompts and determines whether the response should be generated by a general-purpose LLM or by a tool designed for a specific task. This can be implemented using function-calling-aware LLMs. These models are trained to analyze prompts that include a list of function descriptions and return the function name along with the arguments to be called.

Typically, the function to be invoked is described by a JSON object (see example in Fig. 2), which contains the function name, a natural language description, and the definition of the function's parameters. The model then responds with a JSON output containing the selected function name and parameters (both names and values derived from the user prompt).

An example result for the prompt "generate an image of a cat and dog"[8] is shown in Fig. 3.

```
{ "type": "function",
  "function": {
    "name": name,
    "description":
        "Generates image based on
         a user description
         translated into English",
    "parameters": {
      "type": "object",
      "properties": {
        "description": {
          "type": "string",
          "description": "The description
          of an image that user has typed,
          the description has to be in
          English."}
      },
      "required": ["normal_prompt"]}}}
```

Fig. 2. Example of a JSON data block displayed using the verbatim environment.

```
{ "name": "generate_image",
  "parameters": {
    "description": "dog and cat"
  }
}
```

Fig. 3. Examplar response.

B. Chatbot Architecture

The architecture overview is presented in Fig. 4. The chatbot consists of a router, a function-calling LLM (denoted f-LLM), a default LLM, and a set of agents for dedicated functionalities. The router receives input messages, analyzes user questions, and sends them to the function-calling LLM. Based on the response, it directs the query to one of the selected tool agents or to the default LLM.

As mentioned, the chatbot operates in Polish, which presents challenges, since all function-calling datasets used for training LLMs are English-based. Furthermore, Polish models such as Bielik [24] or Qura[9] are not trained for function-calling capabilities. The non-English nature of the conversation necessitates additional efforts, such as translating detected user commands into English (as required by image generation models) and obtaining the correct base forms, which can be particularly challenging for named entities in highly inflected

[7]The link to the chat is not provided due to anonymization requirements; it will be shared if the paper is accepted.

[8]The original prompt was in Polish: "Wygeneruj obraz psa i kota"

[9]https://huggingface.co/OPI-PG/Qra-13b

languages like Polish. For function calling, we utilize Llama 3.1 instruction models [25] that are multilingual, do not generate texts very well in Polish, but understand Polish instructions and are capable of function-calling.

An important aspect of the function-calling LLM module is its ability to respond quickly and properly identify cases where the user prompt is not related to any defined functions. This was achieved through two approaches. First, we implemented a default function call with the description: "If none of the other functions is needed, simply call this" [10].

Second, since the function-calling LLM still has a tendency to answer some user prompts without invoking the default function, we developed a method for rapid detection of such cases. Waiting for the end of the LLM's response can take a long time, especially when the answer is lengthy (as discussed in Section III). To address this, we used the LLM streaming response mechanism. The router analyzes the first token of the response. Since Llama 3.1 prefixes all function call responses with the token <|python_tag|>, it can detect if the model has failed to invoke the default function and is instead responding directly to the user prompt.

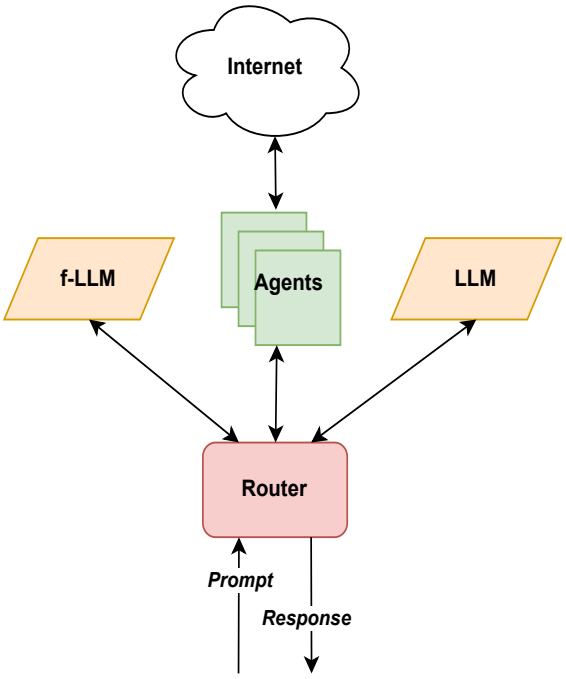

Fig. 4. Diagram of the multi service chatbot.

C. Tool Agents

1) Image generation: The function-calling LLM transforms the user prompt into an English image description (see Fig. 3), allowing us to leverage various image generation models. For this purpose, we utilized the FLUX.1 [schnell] model[11], which features a hybrid architecture that integrates transformer and diffusion techniques [26]. This model was implemented as an additional module within the previously described infrastructure, and the API referenced in Fig. 1 was enhanced with a new endpoint that complies with OpenAI's image generation specifications[12]. Consequently, it includes the functionality to return an image as a URL, which is hosted within our infrastructure for one hour. An example result is illustrated in Fig. 5.

Fig. 5. The exemplar response of FLUX.1 [schnell] model for a prompt "dog and cat".

2) Weather forecast: The weather forecast tool uses the OpenWeather API[13] to obtain JSON encoded information of the current weather in place which name is extracted by the functional-capable LLM. The forecast values obtained are converted into text in Polish by a template and the langauge of the forecats is imporved by the default LLM.

3) Other tools: Wikipedia tool uses the MediaWiki API[14] from obtain information from wikpedia, based on query obtained from function-call LLM. The math tool uses the SymPy [27] a Python library for symbolic mathematics, it allows to evaluate a numercial expression as well as simplifing symbolic one.

D. Problems Encountered

During the implementation of the chatbot, we encountered several challenges related to the quality of function detection by the LLM. Firstly, most of the open source functionally capable LLMs do not support Polish. Consequently, we opted

[10] https://github.com/vllm-project/vllm/issues/7912

[11] https://huggingface.co/black-forest-labs/FLUX.1-schnell
[12] https://platform.openai.com/docs/guides/images
[13] https://openweathermap.org/api
[14] https://www.mediawiki.org/wiki/API

for the Llama 3.1 models. We tested both the 8B and 70B versions of Llama 3.1. The 8B model is significantly faster and requires fewer resources; it can be stored alongside three similar models on a single NVIDIA A100 card, while the 70B model requires at least two such cards. However, the quality of the 70B model is markedly superior to that of the 8B model when it comes to translating image description texts into Polish and accurately obtaining lemmas for named entities in Polish (as used for Wikipedia and weather tools). The 8B model tends to alter Polish letters in the analyzed named entities (for example, "Lądku" is lemmatized as "Lådek"). This raises an important question: should we prioritize speed or quality?

Although the Llama 3.1 70B model yields better results than its 8B counterpart, it still makes occasional errors, such as failing to translate image descriptions into English, which can lead to incorrect image generation.

Another issue is the context of the prompt. Both analyzed Llama 3.1 models tend to lose track of the user's request, specifically, the selection of the appropriate function, as the chat conversation expands. We tried to use the default model to extract user queries from lengthy conversations, but it often failed in many instances. As a result, we implemented a simple workaround: the function-calling mechanism analyzes only the last user prompt. While this approach works well, it sacrifices the ability to maintain context for functional tools, meaning that the chatbot cannot answer questions like, "What is the weather in the city we discussed earlier?". This issue requires further investigation and experimentation with different models and prompts to identify an acceptable solution.

Other challenges were related to the model's tendency to repeat the structure of previous responses. To uphold the rights of authors of the models, APIs, and libraries we utilize, we include information about the sources of the provided data at the end of each tool's response. This is achieved through the use of markup languages and URLs linking to the data or model sources. However, this practice has led to unexpected behavior from the default language model. After a series of responses from tools that include acknowledgments, the default LLM, which answers general questions not addressed by the tool agents, tends to incorporate acknowledgments into its own responses.

Additionally, if an image generation request is not captured by a functionally capable LLM, the default LLM generates fake links to images, following the previous links created by the image tool. Therefore, it is necessary to remove all acknowledgments and links to generated images from prior assistant messages before sending them to the default LLM. These messages must be structured in a way that makes them easily identifiable by specific rules and encapsulated in a manner that is unlikely to be generated by an LLM.

VI. Conclusions

The study examines the effectiveness of three inference approaches for Large Language Models: the standard HuggingFace transformers library, the HuggingFace Text Generation Inference server, and the open-source vLLM library. Experiments conducted with different batch sizes and model sizes demonstrate that vLLM consistently excels compared to the other methods.

Therefore, we have designed and implemented an infrastructure based on the vLLM library for LLM inference as a service. The infrastructure is fully operational and has been generating an average of 4 million tokens per day over the past two months. In addition, a construction of a multi-service chatbot built on the described architecture is presented, illustrating the possibilities of combining LLMs with external APIs or models operating in another modality (image generation). Various detailed problems associated with the implementation of such class systems are discussed, along with the solutions applied to these problems.

Acknowledgements

The work was financed as part of the investment: "CLARIN ERIC – European Research Infrastructure Consortium: Common Language Resources and Technology Infrastructure" (period: 2024-2026) funded by the Polish Ministry of Science and Higher Education (Programme: "Support for the participation of Polish scientific teams in international research infrastructure projects"), agreement number 2024/WK/01.

References

[1] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "Gpt3.int8(): 8-bit matrix multiplication for transformers at scale," in *NeurIPS*, 2022. [Online]. Available: http://papers.nips.cc/paper_files/paper/2022/hash/c3ba4962c05c49636d4c6206a97e9c8a-Abstract-Conference.html

[2] Z. Li, E. Wallace, S. Shen, K. Lin, K. Keutzer, D. Klein, and J. Gonzalez, "Train big, then compress: Rethinking model size for efficient training and inference of transformers," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 2020, pp. 5958–5968. [Online]. Available: http://proceedings.mlr.press/v119/li20m.html

[3] G. Gerganov, "Inference of LLaMA model in pure C/C++," 2023. [Online]. Available: https://github.com/ggerganov/llama.cpp

[4] ——, "GGML - tensor library for machine learning," 2023. [Online]. Available: https://github.com/ggerganov/ggml

[5] J. Bai *et al.*, "ONNX: Open neural network exchange," 2019. [Online]. Available: https://github.com/onnx/onnx

[6] S. Li, H. Liu, Z. Bian, J. Fang, H. Huang, Y. Liu, B. Wang, and Y. You, "Colossal-AI: A unified deep learning system for large-scale parallel training," in *Proceedings of the 52nd International Conference on Parallel Processing*, ser. ICPP '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 766–775.

[7] Y. Liu, S. Li, J. Fang, Y. Shao, B. Yao, and Y. You, "Colossal-Auto: Unified automation of parallelization and activation checkpoint for large-scale models," *arXiv preprint arXiv:2302.02599*, 2023.

[8] S. Li, F. Xue, C. Baranwal, Y. Li, and Y. You, "Sequence Parallelism: Long sequence training from system perspective," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 2391–2404.

[9] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Re, "Flashattention: Fast and memory-efficient exact attention with IO-awareness," in *Advances in Neural Information Processing Systems*, A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, Eds., 2022. [Online]. Available: https://openreview.net/forum?id=H4DqfPSibmx

[10] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

[11] W. Liu, X. Huang, X. Zeng, X. Hao, S. Yu, D. Li, S. Wang, W. Gan, Z. Liu, Y. Yu, Z. Wang, Y. Wang, W. Ning, Y. Hou, B. Wang, C. Wu, X. Wang, Y. Liu, Y. Wang, D. Tang, D. Tu, L. Shang, X. Jiang, R. Tang, D. Lian, Q. Liu, and E. Chen, "ToolACE: Winning the points of LLM function calling," *CoRR*, 2024. [Online]. Available: https://arxiv.org/abs/2409.00920

[12] Y. Qin, S. Liang, Y. Ye, K. Zhu, L. Yan, Y. Lu, Y. Lin, X. Cong, X. Tang, B. Qian, S. Zhao, L. Hong, R. Tian, R. Xie, J. Zhou, M. Gerstein, dahai li, Z. Liu, and M. Sun, "ToolLLM: Facilitating large language models to master 16000+ real-world APIs," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=dHng2O0Jjr

[13] A. Parisi, Y. Zhao, and N. Fiedel, "TALM: tool augmented language models," *CoRR*, vol. abs/2205.12255, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2205.12255

[14] B. Walkowiak and T. Walkowiak, "Implementation of language models within an infrastructure designed for natural language processing," *International Journal of Electronics and Telecommunications*, vol. 70, no. 1, p. 153 – 159, 2024. [Online]. Available: https://www.ijet.pl/index.php/ijet/article/download/10.24425-ijet.2024.149525/1194

[15] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for Transformer-Based generative models," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. Carlsbad, CA: USENIX Association, Jul. 2022, pp. 521–538. [Online]. Available: https://www.usenix.org/conference/osdi22/presentation/yu

[16] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

[17] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6

[18] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "GPTQ: accurate post-training quantization for generative pre-trained transformers," *CoRR*, vol. abs/2210.17323, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2210.17323

[19] P. Zhang, G. Zeng, T. Wang, and W. Lu, "TinyLlama: An open-source small language model," *arXiv preprint arXiv:2401.02385*, 2024.

[20] G. Wang, S. Cheng, X. Zhan, X. Li, S. Song, and Y. Liu, "OpenChat: Advancing open-source language models with mixed-quality data," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=AOJyfhWYHf

[21] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. Xing, H. Zhang, J. E. Gonzalez, and I. Stoica, "Judging LLM-as-a-judge with MT-bench and chatbot arena," in *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. [Online]. Available: https://openreview.net/forum?id=uccHPGDlao

[22] S. Lin, J. Hilton, and O. Evans, "TruthfulQA: Measuring how models mimic human falsehoods," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, S. Muresan, P. Nakov, and A. Villavicencio, Eds. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 3214–3252.

[23] S. Ramírez, "FastAPI - framework, high performance, easy to learn, fast to code, ready for production," 2024. [Online]. Available: https://github.com/fastapi/fastapi

[24] K. Ociepa, L. Flis, R. Kinas, A. Gwozdziej, K. Wrobel, SpeakLeash Team, and Cyfronet Team, "Bielik-11b-v2.3-instruct model card," 2024. [Online]. Available: https://huggingface.co/speakleash/Bielik-11B-v2.3-Instruct

[25] A. Grattafiori *et al.*, "The llama 3 herd of models," *CoRR*, 2024. [Online]. Available: https://arxiv.org/abs/2407.21783

[26] P. Esser, S. Kulal, A. Blattmann, R. Entezari, J. Müller, H. Saini, Y. Levi, D. Lorenz, A. Sauer, F. Boesel, D. Podell, T. Dockhorn, Z. English, and R. Rombach, "Scaling rectified flow transformers for high-resolution image synthesis," in *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=FPnUhsQJ5B

[27] A. Meurer *et al.*, "SymPy: symbolic computing in python," *PeerJ Computer Science*, vol. 3, p. e103, Jan. 2017. [Online]. Available: https://doi.org/10.7717/peerj-cs.103