INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2024, VOL. 70, NO. 1, PP. 153–159
Manuscript received November 23, 2023; revised March, 2024. DOI: 10.24425/ijet.2024.149525

# Implementation of language models within an infrastructure designed for Natural Language Processing

Bartosz Walkowiak, and Tomasz Walkowiak

***Abstract*—This paper explores cost-effective alternatives for resource-constrained environments in the context of language models by investigating methods such as quantization and CPU-based model implementations. The study addresses the computational efficiency of language models during inference and the development of infrastructure for text document processing.**

**The paper discusses related technologies, the CLARIN-PL infrastructure architecture, and implementations of small and large language models. The emphasis is on model formats, data precision, and runtime environments (GPU and CPU). It identifies optimal solutions through extensive experimentation.**

**In addition, the paper advocates for a more comprehensive performance evaluation approach. Instead of reporting only average token throughput, it suggests considering the curve's shape, which can vary from constant to monotonically increasing or decreasing functions. Evaluating token throughput at various curve points, especially for different output token counts, provides a more informative perspective.**

***Keywords*—language model deployment; quantization; Llama-2; E5 model; ONNX; llama.cpp; CLARIN-PL**

## I. Introduction

ARTIFICIAL Intelligence is nowadays the focus of interest of researchers and commercial developers. Many of those interested do not have access to many GPU units needed to run large language models like GPT-3 [1] or Llama-2 [2]. Methods such as quantization and model implementations using the CPU instead of the GPU come to the rescue. However, the question arises as to how much it costs to reduce the model to run on limited resources.

CLARIN-PL[1] is the Polish branch of the CLARIN ERIC[2] research infrastructure. As part of an international project, it must follow trends in AI and use the latest technologies in the field of NLP. To deliver these state-of-the-art technologies, efficient and scalable tools are used.

The article discusses two problems: the computational performance of language models in inference and the building of infrastructure for processing text documents by langauge models.

We introduce here a distinction between large and small language models (LLM and SLM). By LLM, we refer to generative models of general purpose like GPT-3 or Llama-2. By SLM we mean classical transformer-based models such as BERT [3] and T5 [4]. SLM are characterized by their smaller size but also their specialization, as they are fine-tuned for specific tasks.

The paper is organized as follows: Section II discusses technologies and research connected with the topics covered in this article. Section III provides a closer look at the architecture and technologies used in CLARIN-PL. Section IV then describes how the SLM and LLM models could be implemented. The implementation methods are validated through experiments performed on various aspects, including model formats, data precision formats, and runtime environments (GPU and CPU). The next section, Section V, is dedicated to the technical aspects of implementing the models, specifically the techniques used to deploy the models as services.

## II. Related works

A very popular and well-known public repository with many language models is Huggingface. This repository together with the comprehensive Transformers library [5] it supports can be used to process all the most popular SLMs: BERT [3], T5 [4], RoBERTa [6], BART [7], and also LLMs: GPT-3 [1] and Llama-2 [2]. The Transformers library can be used for an inference, as well as for teaching models. The library supports several learning frameworks, with two of the most popular: PyTorch and TensorFlow. Models can also be converted to ONNX [8], an open-source machine-independent standard, or to GGML [9], which enables large models and high performance on commodity hardware.

Another issue is the runtime environment, the model can use the GPU and benefit from its high performance. However, GPU resources are very often limited, and the ability to run only on the CPU is important. An excellent example of CPU inference supporter is llama.cpp [10], which uses GGML format and with which models such as Llama-2, among others, can be used without access to the GPU.

There are production inference servers widely used by industry like Triton Inference Server [11] and TensorFlow

Financed by the European Regional Development Fund as a part of the 2014-2020 Smart Growth Operational Programme, CLARIN - Common Language Resources and Technology Infrastructure, project no. POIR.04.02.00-00C002/19.

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

[1] https://clarin-pl.eu/

[2] https://clarin.eu

Serving [12], but they are general purpose servers for deep model execution and do not have model-specific optimizations. Recently, several serving systems are proposed to optimize Transformer-based LLMs [13]–[17].

Another important model-running technique that is gaining interest and development is quantization [18], [19], or more generally, the choice of model precision. What precision we choose affects, on the one hand, how much graphics card memory we will need, that is, in extreme cases, the possibility of running the model. On the other hand, precision, especially in the case of LLM, affects the quality of the results.

The basic precision of the model involves the use of the float32 type so the values are represented by 32 bits, divided between exponent, mantissa and sign. Other float types used include float16, but also bfloat16 (brain floating point), which differ in the ratio of exponent and mantissa length from the standard float16. Going even lower in precision, choices include float8 E4M3 (4-bit exponent and 3-bit mantissa) and float8 E5M2 (5-bit exponent and 2-bit mantissa). The difference in the way the bits are distributed between exponent and mantissa is the realization of the problems of range (exponent) versus precision (mantissa), because the number of values represented is constant and it is up to us to decide how large a range they will be distributed over.

Typically, models are learned and distributed with a precision of 32 bits and can be quantized in two ways. The first is on-the-fly quantization (implemented by the Transformers library [5]), and the second way is to convert the model beforehand and load the already quantized model (the method used in llama.cpp [10]).

Numerous open-source workflow engines are available for general data pipelining purposes. Apache Airflow [20] or Prefect [21] stand out as popular choices. However, in the field of NLP, the primary focus revolves around libraries such as spaCy [22] and Stanza [23]. Within the context of the CLARIN [24], three solutions have emerged: WebLicht [25], CLARIN-BE [26], and the one detailed in the next chapter.

## III. CLARIN-PL Language Technology Center

The aim of CLARIN is to develop and maintain a framework that facilitates the exchange, application, and enduring availability of linguistic resources and NLP tools to support research in the fields of humanities and social sciences. The CLARIN-PL Language Technology Center (LTC) [27] realizes this goal in Poland, providing effective and user-friendly NLP tools for the social sciences and humanities. This requirement is fulfilled with a complete processing pipeline of the language tools available via the REST API and web-based interface. The solution ensures flexibility and low processing times. Some tools use models that take longer to load than to process the input file, in which case the tool is run as a service with the required models preloaded into memory [28].

CLARIN-PL LTC is based on the microservice architecture [29] that isolates individual services from each other, so each service runs its own process. Services are interconnected by two techniques, one is the message broker, which is RabbitMQ [30], and the other is the shared file system. The broker is responsible for message queueing and is also used for easy horizontal scaling, which allows the system to dynamically adapt to the current load. The message communication mechanism is lightweight, using only basic commands such as; start, finish, and progress of tasks. This mechanism is implemented using the AMQP protocol [31]. AMQP was not used for data exchange because the size of the input/output data can be very large, which can block message queues. That is why the shared file system is used.

This architecture is deployed on a Kubernetes [32] cluster instance. This solution allows for easy containerization, each NLP tool being deployed as a single pod. The automatic scaling of the number of pods is provided by KEDA HPA [33], which scales up or down the number of instances of a given tool depending on the length of the queue of pending jobs on the broker. The entire infrastructure is implemented using the HELM chart [34] tool, which simplifies and automates the generation of multiple similar services.

Implementing a new NLP worker is possible in languages such as Python, Java, and C++ and is based on a library that is responsible for communication. The developer has to implement functions that process the data (the input/output path and input options are passed as parameters to this function).

The orchestration of related tools in the processing chain is provided by the Language Processing Management Notation [35], which enables improved performance through the use of parallel processing. This means that the processing of a corpus (usually provided as a zip archive) can be parallelized between multiple instances of a given tool. Furthermore, the number of instances scales horizontally, so the response to a high load is to increase the number of processing nodes.

In summary, CLARIN LTC provides a configurable and powerful tool that, using well-known technologies and automation mechanisms, enables the launch of more than 100 different language tools.

## IV. Language Models Deployment

As mentioned in the Introduction, contemporary NLP is based on small and large language models, especially transform-based ones. Such models require large computing resources, especially graphics cards are an important aspect. Since hardware resources are always limited, there is a question how to effectively implement such models. Therefore, we performed a set of numerical experiments with a classifier built on a deep language model (Section IV-A) and a chat-like generative large language models implemented (Section IV-B) in different ways and on GPU and CPU.

### A. Small langauge models

The small language models (like BERT or RoBERTa) could be implemented directly by Transformers library or using ONNX format, in both cases they can be run on CPU or GPU. In GPU tests we used NVIDIA GeForce RTX 3090 GPU with 24GB, whereas in CPU tests we used AMD Ryzen Threadripper PRO 3955WX 16-Core processor (4.4 GHz). For

evaluation we used the E5 base model [36]. It is a general-purpose, multilingual sentence embedding model. The model was initialized from xlm-roberta-base and continually trained on a mixture of multilingual datasets in a contrastive manner with a weak supervision. We tested the effectiveness of the model inference for sequences of 512 tokens, and for different size of batches (ranging from 1 to 8). The experiments were carried out in 200 iterations and the results were subsequently averaged. The summarized findings can be observed in Table I.

TABLE I: E5 [36] inference time per task in ms for different deployments on the GPU (NVIDIA GeForce RTX 3090) and CPU (AMD Ryzen Threadripper PRO 3955WX 16-Core). In case of CPU we tested different number of used cores (4, 8, and 16). The reported time is a time to process one text, i.e., in case of batches, it is a time to process one batch divided by its size.

| | Transformers | | | | ONNX | | | |
|---|---|---|---|---|---|---|---|---|
| batch size | 1 | 2 | 4 | 8 | 1 | 2 | 4 | 8 |
| GPU | 12.8 | 8.5 | 8.1 | 7.5 | 9.4 | 8.3 | 7.4 | 6.7 |
| CPU (4) | 423 | 414 | 467 | 468 | 290 | 262 | 266 | 256 |
| CPU (8) | 214 | 196 | 225 | 225 | 184 | 153 | 154 | 147 |
| CPU (16) | 152 | 143 | 158 | 158 | 132 | 111 | 109 | 102 |

The results show that the ONNX implementation outperforms Transformers in terms of inference speed for both CPU and GPU scenarios. However, the difference is significantly larger in the CPU context. Furthermore, the results show that the use of batches accelerates the average inference time, but the effectiveness of batch usage is limited. Therefore, it is crucial to choose the batch size carefully.

### *B. Large Language Models*

Large generative language models have become an essential component of many text-based user interface applications. However, their size (number of weights) is much larger than that of typical language models, ranging from 7 billion to 70 billion weights in case of Llama-2 [2]. Additionally, their typical usage relies on autoregressive generation of responses, wherein the network takes as input the query and the preceding responses generated by the network itself and produces the next token based on the estimated probability. This process iterates, generating successive output tokens. Due to this operational mechanism and the number of weights, the response generation process is time consuming and strongly influenced by the length of the output text.

Therefore, it is crucial to implement such models effectively in production systems. To assess the feasibility of implementing Large Language Models (LLMs), a series of numerical experiments were conducted using the Llama 7B Chat model provided by Meta. The model was converted into the Transformers library format and implemented in three different formats: a 32-bit float, a 16-bit float, and a quantized 8-bit integer.

In the initial step, the GPU inference time was analyzed with respect to response length. All experiments were carried out in identical setups. They were as follows; maximal number of tokens 1024, top_p 0.5, temperature 1, top_k 50, and repeat penalty 1.1. Twenty-five diverse queries were developed, ranging from straightforward ones regarding the number of animal legs to complex biographies of historical figures. Each query was repeated 10 times and the results are presented in Figure 1.

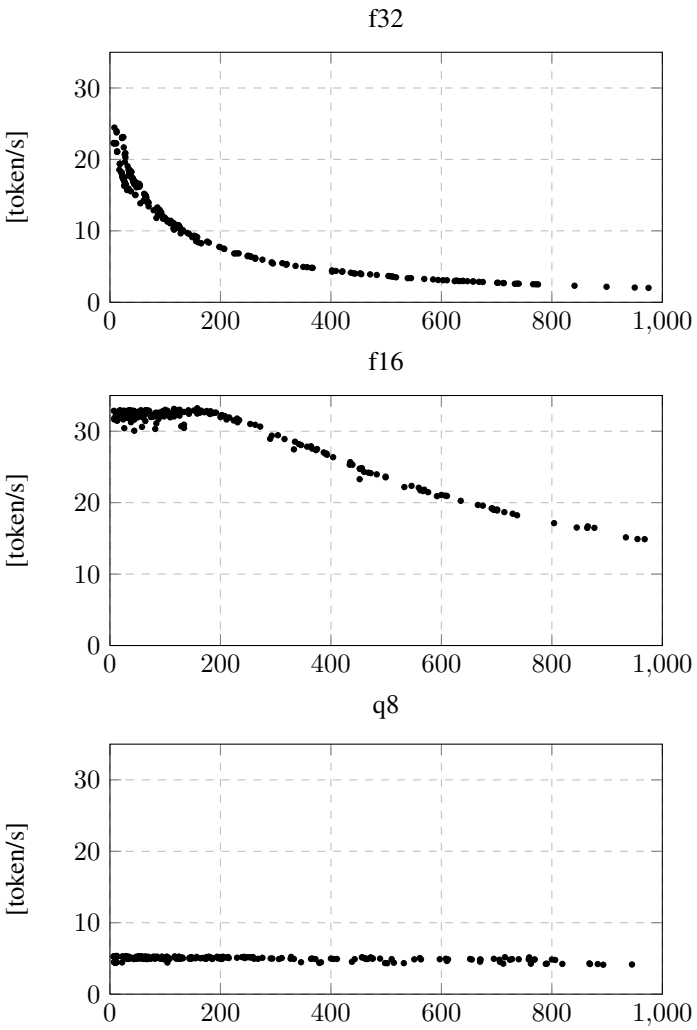


Fig. 1: Llama GPU inference (model Llama-2-7b-chat in Transformers library format). Speed of token generation as a function of generated text size for various number representations in the Transformers library. The experiments were conducted on an NVIDIA A100 80GB GPU.

The first noticeable aspect is the distinct characteristics of the curves. In the f32 format, the curve takes on a logarithmic shape, whereas in the case of f16, we observe a consistent rate of text generation for texts up to approximately 200 tokens, followed by a logarithmic decrease. On the contrary, the speed of text generation remains constant for q8 format, regardless of the length of the generated text. This difference is likely attributed to implementation variances.

A surprising finding, although consistent with other studies [37], is that 8-bit quantization yields worse results compared to 16-bit or even 32-bit floating point models. On the basis of

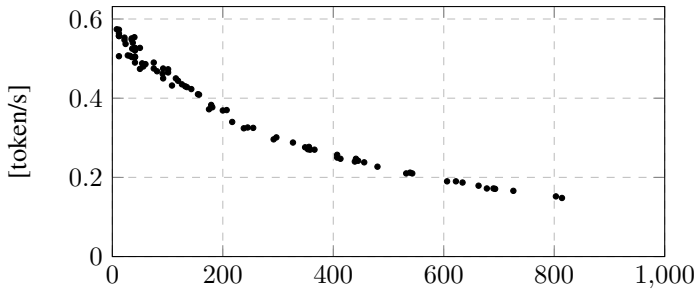


Fig. 2: Speed of token generation as a function of generated text size for Llama model running on CPU and implemented in Transformers library in f32 format. The experiments were carried out on an AMD Ryzen Threadripper PRO 3955WX 16-Core processor (4.4 GHz).

the conducted experiments, it is recommended to implement the f16 model on the GPU.

The next experiment evaluated the performance of the LLama model on the CPU using the default implementation in the Transformers library, which utilizes 32-bit floats. These experiments, similar to the previous case, were conducted on a computer equipped with a 16-core CPU. The results are presented in Figure 2 indicating very long text generation times. For instance, generating a text of 540 tokens would take more than 43 minutes. It shows the impracticality of such an implementation.

Next, we investigate the use of the llama.cpp library [10], which leverages the GGML format [9] to execute large-language models on the CPU. We explored formats similar to those employed in the Transformers library. We also tried to analyze the results for the q4 models, but the quality of the result obtained from the model was not acceptable. The generated texts were unreadable, consisting of tokens from different languages. The results (for formats f32, f16 and q8) are presented in Figure 3. As we can observe, the text generation speeds surpass the implementation in the Transformers library when running on the CPU (2). The shape of the curves remains independent of the format used. An exponential increase in speed is noticeable for short texts, after which the speed stabilizes and remains nearly constant regardless of the length of the text. A closer examination of the code and time dependencies revealed that the generation time comprises two components: model loading (with a constant time factor) and the token generation process with a constant speed. The process of loading the model cannot be omitted during text generation because the model retains context information and has to be reloaded. This phenomenon does not occur with the Transformers library. The best results, characterized by the highest speed, were achieved with the q8 format. Therefore, it is recommended for CPU implementations.

An interesting phenomenon, unrelated to the processing time, is the length of the generated texts. Despite using the same settings in the text generation process and the same queries, this llama.cpp implementation generates significantly shorter texts than the Transformers library. This can be seen by observing the very rare points on the right sides of the graphs in Figure 3, much rarer than in Figures 1 and 2.

An important aspect to note is the variation of speed as a function of the length of the output text, so comparisons of model performance should be reported for a few selected output text length ranges. Therefore, the summary of Llama-2 performance results presented in Table II shows the average speed across four slots for texts of various lengths: 5-20, 100-200, 400-500, and 800-1000. In the case of the llama.cpp implementation, we do not report results for the last slot due to the limited number of results in this range, as discussed earlier. It is worth noting that the best GPU implementation (Transformers f16) is only 2.74 times faster (for longer texts) to 5.13 times faster (for texts up to 20 tokens) than the llama.cpp q8 implementation. This difference is smaller than in the case of BERT models, where the processing time on the GPU differs 11-21 times compared to the CPU time (16 core cases).

TABLE II: Llama inference (model Llama-2-7b-chat) for different implementations: Transformer (Trans.) and llama.cpp, different fomat of weigths (f32, f16, and q8), and running on CPU (MD Ryzen Threadripper PRO 3955WX 16-Core processor at 4.4 GHz and GPU (NVIDIA A100). The table presents an average speed across four slots for texts of various lengths: 5-20, 100-200, 400-500, and 800-1000. We also present the average usage of memory (in case of CPU running models this is a computer RAM, in case of GPU running models this is VRAM.

| Model implement. | Speed [token/s] | | | | Mem [GB] |
|---|---|---|---|---|---|
| | 5-20 | 150-200 | 400-500 | 800- | |
| Trans. f32 (GPU) | 21.56 | 8.50 | 4.13 | - | 31.1 |
| Trans. f16 (GPU) | 32.20 | 32.66 | 24.59 | 16.03 | 18.5 |
| Trans. q8 (GPU) | 4.92 | 5.09 | 4.93 | 4.36 | 8.5 |
| Trans. f32 | 0.54 | 0.38 | 0.24 | - | 26.5 |
| llama.cpp f32 | 1.99 | 2.63 | 2.66 | - | 25.8 |
| llama.cpp f16 | 3.51 | 5.02 | 5.09 | - | 13.2 |
| llama.cpp q8 | 6.27 | 8.93 | 8.96 | - | 7.4 |

## V. Implementing Language Models

### A. *Infrastructure remarks*

In CLARIN-PL, as elsewhere, it is important to manage a limited resource such as the GPU, especially one with high computing power. In the CLARIN LTC case, the problem was raised by the number of available resources, not their power. Kubernetes allows GPU resources to be assigned to a single pod. The solution we used was techniques such as Multi-Instance GPU (MIG) [38] and Time Slicing [39], both of which are supported by the NVIDIA A100 cards. MIG helps when there are not enough GPU instances. It is a technique that logically divides the graphics card, and the card can be split into different parts. The available parts can be 10, 20, 30, or even 40 GB of memory, allowing for a lot of flexibility. The second method allows the GPU to be oversubscribed, but this has the disadvantage of not isolating between resource-using

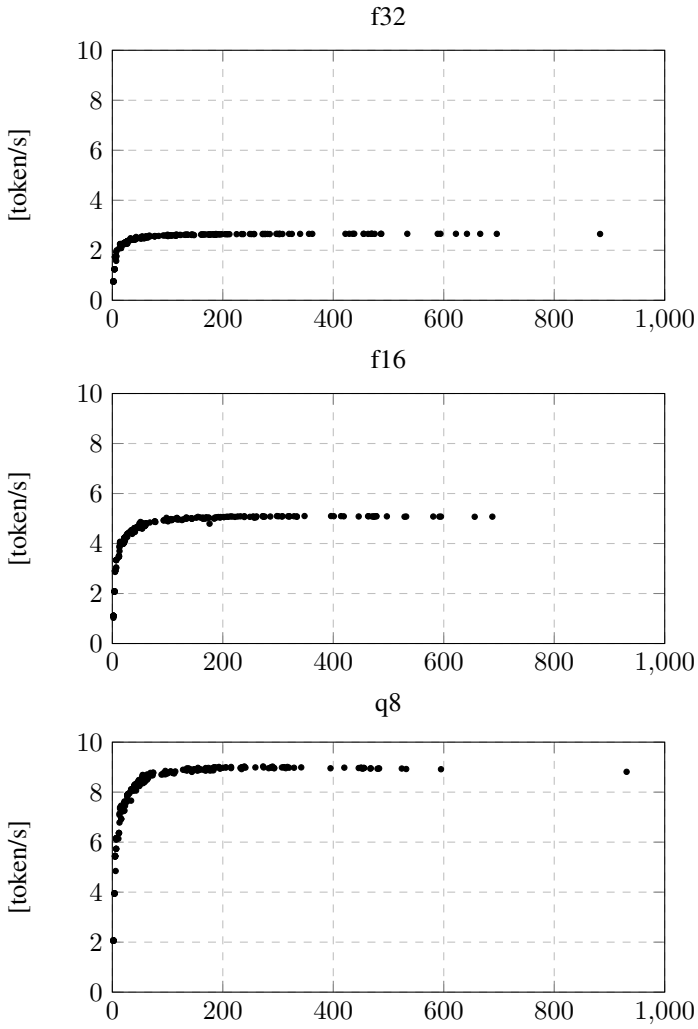


Fig. 3: Llama CPU inference (model Llama-2-7b-chat in GGML format). Speed of token generation as a function of generated text size for various number representations in the llama.cpp library. The experiments were conducted on an AMD Ryzen Threadripper PRO 3955WX 16-Core processor (4.4 GHz).

services; one service can starve others. For the time being, these technologies also have the major disadvantage of not being observable in terms of memory consumption rates.

The techniques described in Section II for dealing with GPU limitations are extended by the technique of implementing several models in a single pod. This implementation bypasses the limitation that one GPU resource can be allocated to one pod. This technique, combined with the memory release mechanism, contributes to efficient memory usage. It is worth noting that models running in a single pod are separated from each other because they are run in separate processes.

### *B. Small Language Models as a Service*

In CLARIN LTC we have implemented models such as T5 [4] used for keyword identification [40] and text devulgarisation [41], but also Sentence-BERT [42] and E5 [36] for embedding generation. The important thing about these models is that they process text of limited length (typically up to 512 tokens), this strong certainty was used in the design of the architecture. The assumption of a small size also applies to the resulting files; an embedding vector of 512 or 768 (embedding), text of a similar size to the input (devulgarisation), or a keyword list limited to a maximum of five (keyword detection). Based on these facts, communication with the services is done via the AMQP protocol, but unlike regular CLARIN-PL workers, the transmission of input/output data is also done via the AMQP protocol (rather than via a shared drive). As can be seen, the service architecture is strictly tailored to the characteristics of the language model and uses their features to operate effectively. See the list of services in Table III.

TABLE III: Small lanaguge models deployed in CLARIN-PL LTC

| service name | task |
|---|---|
| sbert-distiluse-base-multilingual-cased-v1 [42] | embeddings generation |
| sbert-paraphrase-multilingual-mpnet-base-v2 [42] | embeddings generation |
| e5-multilingual-e5-base [4] | embeddings generation |
| t5-voicelab-vlt5-base-keywords [40] | keywords determination |
| t5-DEPOTxT5-base [41] | devulgarization |
| t5-utterance-rewriting-v2-plt5-large | paraphrasing |
| t5-plt5-large-poquad-dst-v2 | dialogue state tracking |

The conversion of the models to the ONNX format was carried out using the HuggingFace Optimum[3] library. However, when using this library, it should be noted that it may not include some elements of the conversion. For example, in the case of Sentence-BERT, elements such as the pooling layer or the additional dense layer are omitted from the Optimum library and had to be additionally stored in PyTorch format. For T5 models, on the other hand, prompts and additional parameters, often used in this type of models, must be stored separately.

In addition to the limitations to GPU access, the memory of graphics cards is also not infinite. To make better use of the available memory, a memory release mechanism has been introduced. It works as follows, when a model is not used for a certain period (the length of this period can vary from model to model), then the model is removed from the card's memory and reloaded when it is needed again.

We are using the KEDA HPA [33] automatic scaling features to scale up or down the number of service instances of a given model depending on the length of the queue of pending jobs on the RabbitMQ broker.

SLM models can be used inside CLARIN-PL services (Section III) via the AMQP protocol (the service library has been extended with dedicated functions) as well as outside the system via the REST interface.

[3] https://huggingface.co/docs/optimum

### C. Generative Deep Model as a Service

We have also implemented large language models such as Llama-2 [2] and RWKV [43]. Their implementation is very similar to what was mentioned for SML, but it needs to be adapted to the specifics of LLMs. The main difference is that LLM models running in chat mode require context and, therefore, access to the conversation history. To achieve this, the worker's input includes the conversation history in JSON format. Additionally, the requests sent to the model have been expanded to include parameters such as temperature and top_p, as well as the ability to choose between two response modes. In single mode, the response is returned when the entire response is generated. In streaming mode, on the other hand, parts of the response (tokens) are returned as they are generated. The LLM services are exposed to the outside world as REST services[4], and for the streaming mode, we utilize the Server Side Events mechanism [44].

People are also direct users of LLMs, so we built a dedicated application with a web-based user interface (Figure 4) to communicate with the services. Since an LLM running in chat mode requires context and access to the conversation history, we implemented an infrastructure in which each conversation thread is stored in a separate record in the database. This approach allows for fast access and ensures security.

The user interface allows the model to be used in two modes: chat mode and instruction mode. In chat mode, the model has access to the conversation history, and in instruction mode, the input is split into an instruction and text, allowing for specific tasks to be performed. When chat mode is used, the client that continues the conversation in the thread sends a new input, thread ID, and parameters. The conversation context, retrieved from the database, is prefixed to the new input, and such a query with parameters is passed to the model. The application interface for the different models is unified; regardless of whether Llama-2 or RWKV is selected, the requests sent are the same, differing only in the "model" parameter in the request body.

## VI. Conclusions

As demonstrated in Section II, there are numerous implementations of language models to choose from, together with various techniques to support model deployment. We conducted tests on some of these implementations to run models in inference mode on both GPU and CPU. In addition, we explained the differences and characteristics of runtime environments and precision formats. These results enabled us to identify the most suitable solutions.

We showcased the implementation of small language models at the CLARIN-PL Language Technology Center. This approach relies on the ONNX format and utilizes AMQP for inter-service communication, facilitating efficient and scalable integration with other CLARIN-PL services. For LLM, we employed a Transformer 16-bit implementation for GPU and GGML 8-bit for CPU. Furthermore, we provided a demonstration of the user interface for auto-generative language models.

[4]https://services.clarin-pl.eu/api/v1/docs

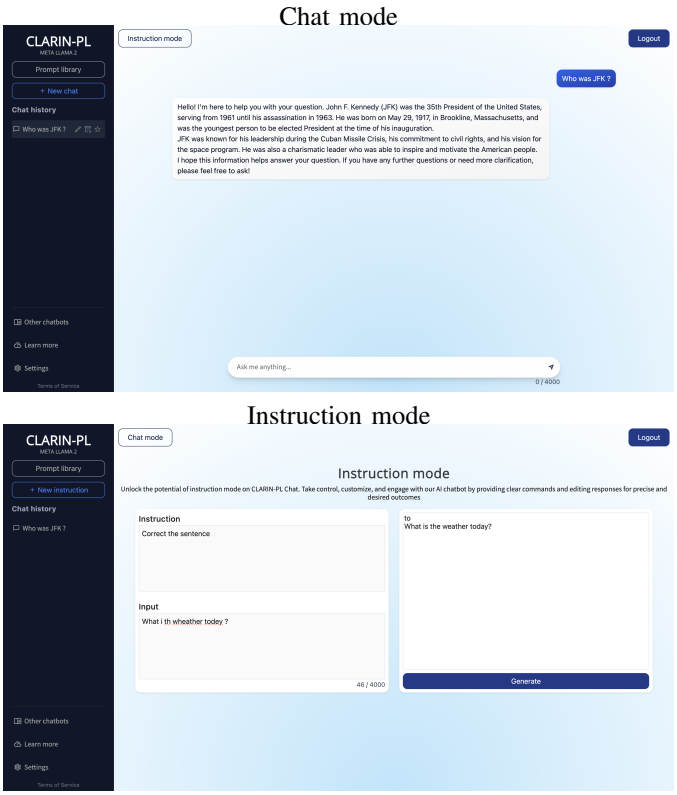


Fig. 4: CLARIN-PL chat interface (https://chat.clarin-pl.eu/)

The smallest precision we encountered involved storing information using only 4 bits, imposing significant size limitations and resulting in substantial information loss.

The phenomenon of observing different shapes in the curves during experiments leads to the conclusion that reporting only the average speed of tokens per second, which has been the standard practice in many previous works [10], [17], [18], is not entirely informative. This is because the curve can exhibit various shapes, including constant, monotonically increasing, or monotonically decreasing functions. A more effective approach is to report the average number of tokens per second at several points along the curve, specifically for low, medium, and high numbers of output tokens.

## References

[1] T. B. Brown et al., "Language models are few-shot learners," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS'20. Red Hook, NY, USA: Curran Associates Inc., 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[2] H. Touvron et al., "Llama 2: Open foundation and fine-tuned chat models," 2023. [Online]. Available: doi:10.48550/arXiv.2307.09288

[3] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Association for Computational Linguistics, 2019, pp. 4171–4186. [Online]. Available: doi:10.18653/v1/n19-1423

[4] C. Raffel et al., "Exploring the limits of transfer learning with a unified text-to-text transformer," *J. Mach. Learn. Res.*, vol. 21, p. 67, 2020, id/No 140. [Online]. Available: jmlr.csail.mit.edu/papers/v21/20-074.html

[5] T. Wolf *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, 2020, pp. 38–45. [Online]. Available: https://aclanthology.org/2020.emnlp-demos.6

[6] Y. Liu et al., "RoBERTa: A robustly optimized BERT pretraining approach," 2019. [Online]. Available: http://arxiv.org/abs/1907.11692

[7] M. Lewis *et al.*, "BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, Jul. 2020, pp. 7871–7880. [Online]. Available: doi:10.18653/v1/2020.acl-main.703

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

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

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

[11] NVIDIA Corporation, "Triton inference server: An optimized cloud and edge inferencing," 2019. [Online]. Available: https://github.com/triton-inference-server/server

[12] C. Olston *et al.*, "Tensorflow-serving: Flexible, high-performance ml serving," in *Workshop on ML Systems at NIPS 2017*, 2017. [Online]. Available: doi:10.48550/arXiv.1712.06139

[13] NVIDIA Corporation, "Fastertransformer," 2019. [Online]. Available: https://github.com/NVIDIA/FasterTransformer

[14] D. Li, H. Wang, R. Shao, H. Guo, E. P. Xing, and H. Zhang, "MPCFORMER: fast, performant and provate transformer inference with MPC," in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. [Online]. Available: https://openreview.net/pdf?id=CWmvjOEhgH-

[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] N. Yang *et al.*, "Inference with reference: Lossless acceleration of large language models," 2023. [Online]. Available: doi:10.48550/arXiv.2304.04487

[17] B. Wu, Y. Zhong, Z. Zhang, G. Huang, X. Liu, and X. Jin, "Fast distributed inference serving for large language models," *arXiv:2305.05920*, 2023.

[18] 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

[19] 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

[20] The Apache Software Foundation, "Apache Airflow," 2023. [Online]. Available: https://airflow.apache.org

[21] Prefect Technologies, Inc., "Prefect," 2023. [Online]. Available: https://www.prefect.io

[22] Explosion, "spaCy: Industrial-strength NLP," 2023. [Online]. Available: https://github.com/explosion/spaCy

[23] P. Qi, Y. Zhang, Y. Zhang, J. Bolton, and C. D. Manning, "Stanza: A python natural language processing toolkit for many human languages," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Online: Association for Computational Linguistics, Jul. 2020, pp. 101–108. [Online]. Available: doi:10.18653/v1/2020.acl-demos.14

[24] A. Branco *et al.*, "The CLARIN infrastructure as an interoperable language technology platform for SSH and beyond," *Language Resources and Evaluation*, Jun. 2023. [Online]. Available: doi:10.1007/s10579-023-09658-z

[25] M. Hinrichs, T. Zastrow, and E. W. Hinrichs, "Weblicht: Web-based LRT services in a distributed escience infrastructure," in *Proceedings of the International Conference on Language Resources and Evaluation, LREC 2010, 17-23 May 2010, Valletta, Malta*, N. Calzolari *et al.*, Eds. European Language Resources Association, 2010. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2010/summaries/270.html

[26] A. Lemmens and V. Vandeghinste, "A lightweight NLP workflow engine for CLARIN-BE," in *CLARIN Annual Conference Proceedings*, 2022, pp. 29–34. [Online]. Available: https://www.clarin.eu/sites/default/files/CLARIN2022_P_2.1.4_LemmensVandeghinste.pdf

[27] M. Pol, T. Walkowiak, and M. Piasecki, "Towards CLARIN-PL LTC digital research platform for: Depositing, processing, analyzing and visualizing language data," in *Reliability and Statistics in Transportation and Communication*, I. Kabashkin, I. Yatskiv, and O. Prentkovskis, Eds. Cham: Springer International Publishing, 2018, pp. 485–494. [Online]. Available: doi:10.1007/978-3-319-74454-4_47

[28] T. Walkowiak, "Web based engine for processing and clustering of polish texts," in *Theory and Engineering of Complex Systems and Dependability*, W. Zamojski *et al.*, Eds. Cham: Springer International Publishing, 2015, pp. 515–522. [Online]. Available: doi:10.1007/978-3-319-19216-1_49

[29] S. Newman, *Monolith to microservices: evolutionary patterns to transform your monolith*. O'Reilly Media, 2019.

[30] VMware, "RabbitMQ," 2023. [Online]. Available: https://www.rabbitmq.com

[31] OASIS, "Advanced Message Queuing Protocol," 2023. [Online]. Available: https://www.amqp.org

[32] The Linux Foundation, "Kubernetes," 2023. [Online]. Available: https://kubernetes.io

[33] ——, "Kubernetes Event-driven Autoscaling," 2023. [Online]. Available: https://keda.sh

[34] ——, "HELM The package manager for Kubernetes," 2023. [Online]. Available: https://helm.sh

[35] T. Walkowiak, "Language processing modelling notation – orchestration of NLP microservices," in *Advances in Dependability Engineering of Complex Systems*, ser. Advances in Intelligent Systems and Computing, W. Zamojski *et al.*, Eds., vol. 582. Springer, 2017, pp. 464–473. [Online]. Available: doi:10.1007/978-3-319-59415-6_44

[36] L. Wang, N. Yang, X. Huang, B. Jiao, L. Yang, D. Jiang, R. Majumder, and F. Wei, "Text embeddings by weakly-supervised contrastive pre-training," 2022. [Online]. Available: doi:10.48550/arXiv.2212.03533

[37] Y. Belkada and T. Dettmers, "A gentle introduction to 8-bit matrix multiplication for transformers at scale using Hugging Face Transformers, Accelerate and bitsandbytes," 2022. [Online]. Available: https://huggingface.co/blog/hf-bitsandbytes-integration

[38] NVIDIA Corporation, "NVIDIA Multi-Instance GPU user guide," 2023. [Online]. Available: https://docs.nvidia.com/datacenter/tesla/pdf/NVIDIA_MIG_User_Guide.pdf

[39] ——, "Time-slicing GPUs in Kubernetes," 2023. [Online]. Available: https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/gpu-sharing.html

[40] P. Pezik, A. Mikołajczyk, A. Wawrzyński, B. Nitoń, and M. Ogrodniczuk, "Keyword extraction from short texts with a text-to-text transfer transformer," in *Recent Challenges in Intelligent Information and Database Systems*, E. Szczerbicki *et al.*, Eds. Singapore: Springer Nature Singapore, 2022, pp. 530–542. [Online]. Available: doi:10.1007/978-981-19-8234-7_41

[41] C. Klamra *et al.*, "Devulgarization of polish texts using pre-trained language models," in *Computational Science – ICCS 2022*. Cham: Springer International Publishing, 2022, pp. 49–55. [Online]. Available: doi:10.1007/978-3-031-08754-7_7

[42] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. [Online]. Available: http://arxiv.org/abs/1908.10084

[43] B. Peng *et al.*, "RWKV: Reinventing RNNs for the transformer era," 2023. [Online]. Available: doi:10.48550/arXiv.2305.13048

[44] L. de la Torre, J. Chacon, D. Chaos, S. Dormido, and J. Sánchez, "Using server-sent events for event-based control in networked control systems," *IFAC-PapersOnLine*, vol. 52, no. 9, pp. 260–265, 2019, 12th IFAC Symposium on Advances in Control Education ACE 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896319305555