A dutch question answering retrieval augmented generation bot

S.R. Peiter[1, a)]
ICTU, PO Box 84011, 2508 AA Den Haag, The Netherlands

(*sarwan.peiter@ictu.nl, sarwan@smartworkz.nl)

(Dated: 1 March 2024)

Large pretrained language models have shown unprecedented ability to understand and generate text and their integration with other data tools further improve and expand their capabilities. However, these model often require significant computational resources to operate effectively and have predominantly been trained on English corpus limiting their proficiency in other languages. Here, we introduce a document question answering bot designed for Dutch language utilization. Using a 8-bit quantized 13 Billion (B) Llama large language model (LLM) and a retrieval augmented generation (RAG) architecture, our system operates efficiently on a consumer-grade setup containing a NVIDIA RTX2070 GPU device. Using this GPU device resulted in approximate 10× increase of token inference rate compared to traditional CPU inference. We evaluated the RAG bot against chatGPT-3.5, and showed a cosine similarity of 0.8 and larger between responses to the same question submitted independently to both systems. These promising results are the first step in using smaller models with a lower computational footprint for document question answering tasks, paving the way for efficient and accessible natural language processing solutions.

Large pretrained language models have gone through significant advancements in recent years, expanding from text-to-text generation[1,2] to tasks like text-to-image[3] and the more recent text-to-video generation[4]. One area of ongoing research focuses on improving their quality and extending their functionalities by integrating them with vector databases and other data tools, facilitating seamless data transfer between the language model and databases. This approach, known as retrieval-augmented generation (RAG)[5,6], enables the conditioning of the language model and provides additional information when needed, resulting in more accurate responses.

The RAG technique proves to be very useful in document question answering, where contextual information from documents is provided ad-hoc to the language model. This method offers convenience as it eliminates the need to train the language model on confidential data, which would typically involve using expensive hardware. In the real world, various scenarios require extracting information for example about medical papers[7,8], or policy and legislative information within policy documents. These are areas where document question answering systems prove to be invaluable. As the size of these language models increase, their accuracy improves, leading to more precise and reliable question answering with fewer errors[9,10].

However, a significant disadvantage are the high computational costs associated with inference, requiring the use of expensive server farms to achieve suitable response times[11,12]. In addition, the cost-performance ratio scales linearly with the size of the model[13]. For instance, the cost difference associated with training GPT-3.5 and GPT-4 is substantial, given the more than tenfold increase in model size[14]. While larger models indeed show better performance, the question arises whether the cost justifies the intended use case.

Another challenge is the language fluency of existing question answering systems, which often rely on open source language models trained predominantly on English corpora[15–17], limiting their effectiveness in other languages. Although closed-source language models may offer better fluency in various languages, they typically restrict access to their models and require payment for their services, making them inaccessible to all users.

In response to these challenges, we present a document question answering bot fluent in Dutch, deployed on local hardware equipped with a consumer-grade NVIDIA RTX2070 GPU device. We implement a RAG architecture using a quantized 13B Llama LLM, enabling partial inference on the local GPU with limited memory capacity. Our evaluation demonstrates that our bot achieves high similarity (> 0.8) when compared to responses from chatGPT-3.5. While these results are promising, there is ample room for improvement to enhance performance further. Potential optimizations include streamlining the document retrieval process, refining prompts, and modifying the RAG workflow. This approach holds promise as it approaches the accuracy of larger models like chatGPT-3.5 while maintaining a significantly lower computational footprint.

I. RAG ARCHITECTURE

Our document question answering bot implements a naive RAG model[5] as shown in Fig.1. In this RAG implementation, we augment the LLM with private contextual data alongside the query, which is crucial since the LLM lacks specific training on our confidential data. Without this contextual information augmentation, the response generated by the LLM may get overly generalized when it comes to the topics covered in the documents.

Using a RAG model is advantageous compared to other techniques such as fine-tuning, where we need a significant

[a)]Machine learning @ Smartworkz

amount of high end hardware resources to further train the LLM. Overall, the RAG implementation tends to outperform fine tuned models for relatively small datasets. This is because certain data and instruction needs to occur several times before the model effectively retains them through fine-tuning[18].

We build a RAG bot on a consumer-grade desktop containing an Intel i9 9900K[19], NVIDIA RTX2700 and 16 gigabytes (GB) RAM. We use a 13B Llama model as our base LLM[16,20] and since this model has been trained on a corpus that contained limited amounts of Dutch, we resort to a Dutch instruction tuned 13B Llama model[21]. The weights of most LLM are represented by 16 bits floating point number[22] and therefore a 13B model requires at least 52 GB of memory space of either RAM or VRAM. However, we do not have this substantial amount of memory to our availability and hence, we make use of a quantized version of the model. Quantization refers to a technique where the precision of some layers within the neural network (NN) weights is significantly reduced, while preserving the model accuracy[23,24]. This is possible, because not all operations related to inference require high precision floating point representation and in some cases we can get away by using lower precision models. We make use of the llama.cpp backend, which is an application that is used to quantize LLM models to lower resolutions and allows inference tasks to be split and run on CPU, GPU or both depending on the user preference. In our setup, we use the 8 bit quantized version, which occupies 9 GB of memory space. This fits comfortably in our RAM, while the VRAM of the GPU can only take up to 56% of the model size. This version of the model have shown to have almost the same perplexity as the 16-bit version of the model[24].

On a higher level the RAG workflow can be divided in three steps.

1. Indexing: This part of the architecture is responsible for indexing and storing the documents of interest that is provided via the user interface (UI) that is implemented with streamlit framework. The contents of the document are not indexed as a whole; instead, they are initially split up in smaller chunks. This is necessary to prevent surpassing Llama (13B) context window size of 4K[16], which sets the limit at which the language model can still effectively process text. Exceeding this limit leads to an increase of inference time and can deteriorate output accuracy and cause possible hallucinations or mid-air drops[25]. While splitting the documents into smaller chunks is vital, we must pay close attention to this process. When done inappropriately, context information between adjacent chunks can get lost. To do this efficiently while still keeping semantic pieces of text related, we use a recursive splitter. We noticed that an appropriate chunk size and chunk overlap depend very much on the type and contents of the documents and so there is no one fits all solution. For the types of documents used in our setup, we choose a chunk size between 100 and 350 tokens, while the chunk overlap is set between 50 and 200. Hereafter, the chunks of text are encoded in vectors using the openAI text-embedding-3-large model and stored in a specialized database called a vector database. We have used chromaDB for this purpose.

2. Retrieval: This stage involves constructing a series of prompts after a user query has been submitted via the UI. The user query is first embedded in a hardcoded prompt template with the instruction to rephrase the original query conditioned on the previous queries obtained from the chat history. The use of chat history as part of the prompt is to enable ongoing dialogue and are therefore stored and obtained from the cache, which is an in-memory data object. Hereafter, the rephrased query gets transcoded using the above mentioned openAI text embedder into a vector representation. The system then continues with calculating the similarity score between the query vector and the vectorized chunks in the database. The top 4 highest ranked chunks are selected and used as context for answering the user query.

3. Generation: In this phase, the prompt is compiled with rephrased user query and relevant context in a hard coded prompt template with instructions on how to answer the user query and is sent to the LLM. Depending on the query, the model may formulate a response from its inherent parametric knowledge or from the information that has been present in the documents. Every prompt that is sent to the LLM is also logged by the logging module. This allows for monitoring LLM performance and user queries essential for debugging.

The langchain framework is used to implement the RAG architecture because it allows for efficient passing data and prompts between the different components within the architecturec[26]. Hence, there is no need to interact with the API of the different components separately, saving us time.

After implementing a RAG architecture for our document question answering bot, we explore the response time of our LLM. This aspect holds equal significance to the response accuracy. A swift response time allows for faster feedback and enhances the overall user experience. Our specified upper bound of the response time lies around 15 to 20 seconds, which is comparable to chatGPT-3.5 response times[27]. In the next section we verify whether our specific implementation meet our requirements.

II. INFERENCE TIME PROFILE

When it comes to time profiling the LLM, we first model the LLM as a system that is composed of a text evaluation and a text generation component as shown in Fig.2a. The first part is concerned with extracting the meaning and context of the query, while the latter part attempts to predict the response word by word depending on a set of tokens that it generates related to the context. When a query containing N_{in} tokens is submitted to the LLM, it undergoes evaluation, where the query is vectorized by dedicated transcoders and then represented as a series of matrices. These matrices are passed to

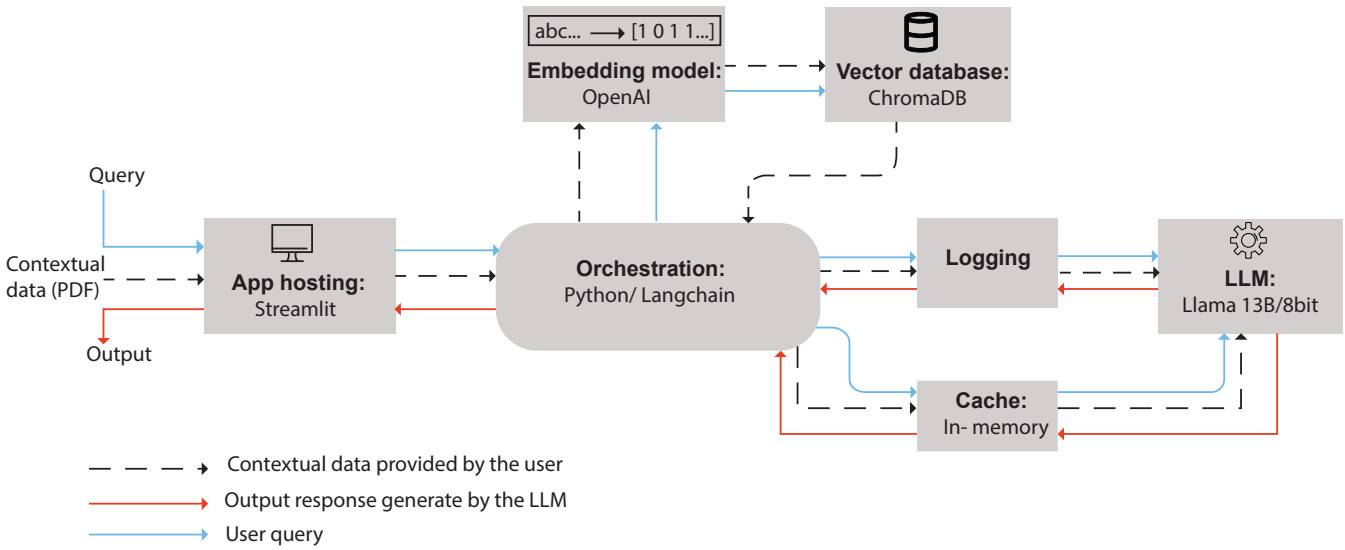

FIG. 1. **Layout of RAG architecture.** It consists of a chromaDB vector database connected to an openAI text embedding model (text-embedding-3-large) for vectorization of contextual data (PDF) provided through a streamlit UI. In-memory caching is added for storing and retrieving historical prompts while the addition of a logging module is used for monitoring application performance, storing a log of prompts and debugging. The Llama 13B LLM receives the user query and proceeds by generating a response conditioned by the contextual data. The pipeline is orchestrated with Langchain in python and is triggered when the user submits a query via the UI built.

the neural network which essentially boils down to a series of matrix multiplication operations. After evaluation, a response is generated containing N_{out} tokens. The output tokens are generated sequentially, with each subsequent token predicted based on the previous one. The total response time of the model thus depends on the number of input N_{in} and output tokens N_{out} and is expressed as

$$R.T = \tau_e \cdot N_{in} + \tau_g \cdot N_{out} \quad (1)$$

We measure τ_e and τ_g for several quantizations of the Llama 7B and 13B model by querying the models with a question containing six tokens and subsequently obtaining a three token response. We show the text evaluation τ_e and generation time τ_g per token for the several quantized models with respect to the number of NN layers exported to the GPU memory in Fig.2b and c respectively.

We notice that τ_e and τ_g follow the same linear trend i.e. they both decrease as more NN layers are exported to the GPU, but τ_e is consistently smaller than τ_g over the entire measured range. The disparity between τ_e and τ_g relative to τ_e for models with similar size and resolutions also widens. For all variants, we observe that τ_e converges to below 10 ms with the 7B/4-bit model having the fastest $\tau_e \approx 3$ ms, while the 7B/8-bit and 13B/4-bit version converge to ≈ 8 ms. Currently the origin of this discrepancy is unclear. However, we did notice that both τ_e and τ_g are negatively impacted if other applications are using the RAM or VRAM, but have not investigated this systematically. Whether this phenomena is the reason behind the observed discrepancy is unknown to us. On the other hand, τ_g converges to ≈ 20 ms for the 7B/8-bit and 13B/4-bit versions, while ≈ 12 ms for the 7B/4-bit version. When running inference solely on the CPU, corresponding to a GPU load of zero, we notice that both τ_e and τ_g increases with the size of the model, which has also been observed in literature[28]. In this scenario, the performance is limited by the CPU and this limitation persists until a GPU load of one is reached.

For our current 13B/8-bit model, we were able to export a maximum of 50% of the NN layers to the GPU before being limited by the 8 GB internal memory of the GPU, as indicated by the green star in Fig.2b,c. We obtain a τ_e and τ_g of ≈ 60 and 220 ms respectively. To derive an upper bound for the total response time using Eq.1, we assume a context window size of 512 tokens and output token limit of 256, resulting in a response time of 87 s. Given the average user query and response containing 25 and 100 tokens respectively, we derive an average response time of 23.5 s. This is higher compared to chatGPT-3.5 response time of 15 s. However, in case of a GPU load of one and assuming a τ_e and τ_g of 3 and 12 ms respectively for our 13B/8-bit model, we find an upper bound and average response time of 4.6 s and 1.3 s respectively. This response time is much more acceptable in a multi user production environment, making the GPU-only setup ideal for achieving short response times. In general, GPUs have better performance at inference than a CPU, because mathematical operations related to inference are implemented very efficiently in a parallel fashion by numerical software that leverages the underlying GPU multi-core hardware architecture[29].

After investigating the performance of our 13B/8-bit Llama model on our local hardware, we proceed with evaluating our document question answering bot in the next section.

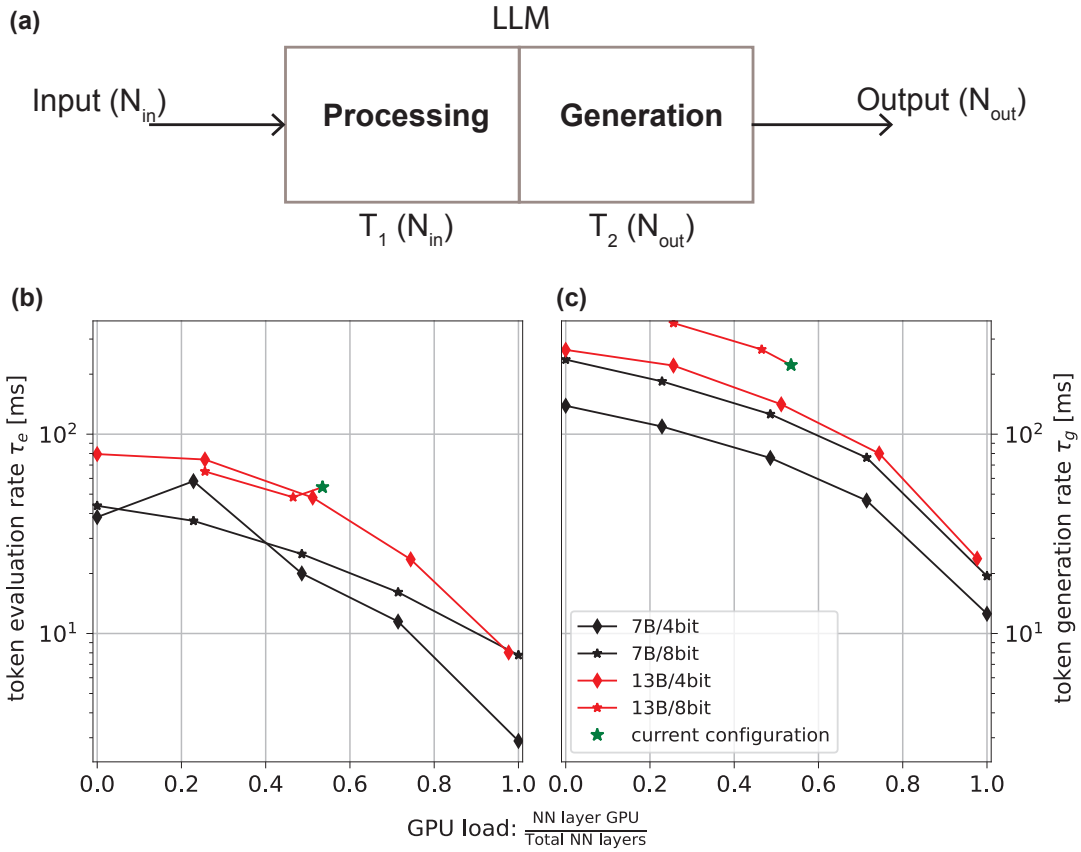

FIG. 2. **Response time characterization of LLM. (a)** A response model of a LLM that is split in an evaluation and generation part. The time spent in the evaluation and generation part depends on the number of tokens and is expressed as $T_1(N_{in})$ and $T_2(N_{out})$ respectively. **(b,c)** Evaluation and generation response times of a Llama 7B and 13B that are quantized to 4 and 8 bits. Both response times decrease as more NN layers are exported to the GPU. The green star reflects our current system configuration i.e. 13B/8-bit.

III. EVALUATION

Evaluation of our RAG system is necessary, because it helps us better understand the strength and weaknesses of the question answering bot, thus giving us insights in the performance of the model. In addition, we get insights in the human-LLM interactions and if necessary improve and fine-tune future interaction design. In addition, we can ensure safety and reliability of our system, which is an important requirement when working mainly with governmental data and systems. But evaluation of a LLM is not so straightforward as we need to ask ourselves what tasks we should perform to be able to evaluate the performance of our system. There is not a singular task that we can perform to gauge the overall performance of our LLM. In literature, there exist numerous evaluation techniques designed to assess the various characteristics of NLP systems, encompassing both understanding and generation[30].

The characteristics that we prioritize during this experimental phase of our RAG bot are natural language understanding, in particular semantic understanding and natural language generation e.g. summarization, grammar and factuality in Dutch. In this article, we limit the scope to evaluating our model on question answering in particular correctness and honesty. We subject the RAG bot to three PDF documents written in Dutch i.e. a boiler manual, a news and a research article. Per document, we formulate a set of three non related questions that we submit to the RAG bot and chatGPT-3.5 independently at almost the same time. The response of chatGPT-3.5 is assumed to be always correct and is used as the ground truth or reference to which the response of our RAG bot is compared against. This is a valid assumption since chatGPT-3.5 excels when it comes to question answering with provided user context and is one of the most accurate LLM as of the time of writing[31] . For each question, we want the responses to be robust, correct and accurate with respect to the reference response. Therefore we resort to the text level similarity metric. We avoid word-level metrics such as precision and recall as this does not capture the overall meaning of the response, but instead quantifies the word similarities between the reference and generated response. We quantify the similarity between the responses with the cosine similarity given by

$$\cos\theta = \frac{A \cdot B}{|A||B|} \tag{2}$$

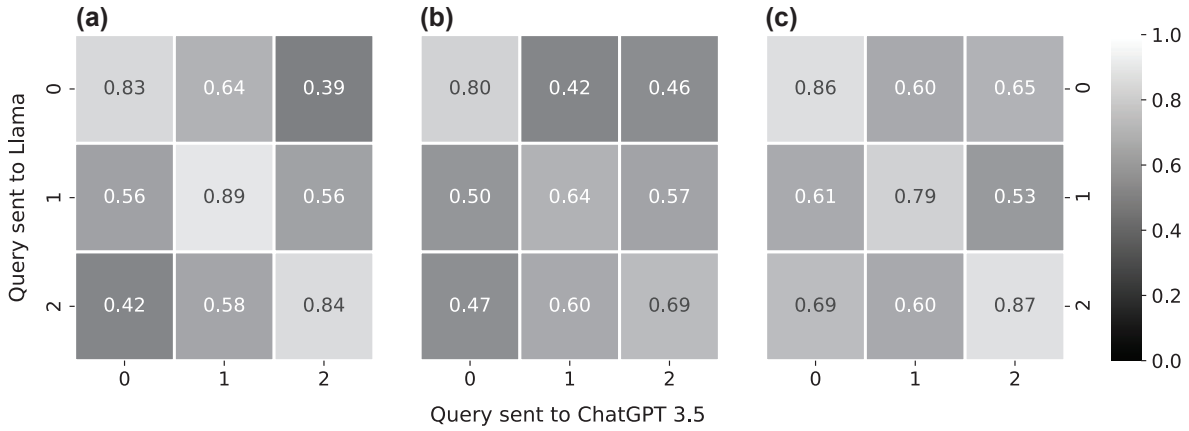

FIG. 3. **Cosine similarity heatmap between response from chatgpt and RAG bot.** Responses are obtained and compared after the queries are submitted to both LLMs. Queries are formulated based on the type of dutch written document i.e. **(a)** a boiler manual, **(b)** a news article and **(c)** a scientific article.

and is shown in Fig.3 for each document. For vectorization of text to calculate cosine similarities, we use OpenAI text-embedding-3-large text to vector embedding, since it is multilingual and one of most accurate models available when it comes to text to vector embeddings. We notice high similarity scores of 0.8 and above in Fig.3a,c for responses to the same question by Llama and chatGPT-3.5 as seen on the diagonal elements of the heatmaps. The off diagonal elements tend to have lower scores, since the questions are not related to one another and therefore the corresponding responses are unalike. In Fig.3b, the diagonal elements related to the second and third question have similarity scores below 0.8 i.e. 0.64 and 0.69, respectively. This discrepancy arises because chunks retrieved from the vector database did not contain the enough information to answer the questions posed to the LLMs accurately. While chatGPT-3.5 responded honestly by admitting its lack of knowledge, our RAG bot generated a response attempting to answer the question regardless. The response was either a literal copy of the context provided with the question or was completely unrelated to the question itself. In addition, we do notice our RAG bot in some cases generating unnecessary long answers or repeating itself whenever it is unable to generate an accurate response. This is contrary to our instructions provided with each question to admit its lack of knowledge whenever it does not know the answer. Our RAG bot simply ignores this instruction, so this behavior needs further investigation to understand why this happens. In addition, we sometimes noticed mid air drops and hallucinations meaning that the response does not relate to the provided context. Strong bias and toxic responses are also some of the challenges we faced in this phase. Nevertheless, these results are promising since in some instances, as shown above, our smaller 13B Llama RAG bot reaches the accuracy of the larger 175B chatGPT-3.5 model. This is advantageous, because with further optimization, we will be able to reach better overall performance on our local setup and do not need to rely on bigger LLM that can only be run on expensive GPU servers.

IV. DISCUSSION

We have set up a document question answering RAG bot that is able to understand and generate text in Dutch. The RAG bot utilizes a dutch fine-tuned 13B Llama model[21] that is quantized to 8 bits such that inference could be run on our local hardware with limited CPU and GPU memory, 16 and 8 GB respectively. Not only does the architecture of the RAG model enable ongoing dialogue made possible by caching, but it also overcomes the limitation of the finite context window size of the LLM when dealing with larger PDF documents. This is achieved by employing a vector database to store the content of provided PDF content in smaller chunks. After profiling the token evaluation τ_e and generation τ_g for the Llama 13B model, we infer that inference is faster by a factor ≈ 10 when performed on the GPU instead of CPU. This performance increase is a result of the underlying hardware architecture that numerical software leverages to implement LLM inference algorithms[29]. Furthermore the accuracy of the RAG bot is evaluated by comparing responses to predefined queries against chatGPT-3.5. We observe cosine similarity scores of 0.8 and higher, which is promising given that the 13B Llama model is smaller by more than an order of magnitude than chatGPT-3.5. However, our RAG system frequently suffers from hallucinations and mid air drops and is therefore not robust and reliable yet. As the reason for this behaviour is still unclear, we suspect that one of the causes might be related to quantization of the LLM. Quantization trades in model resolution for size, which we suspect result in a decrease of inference accuracy. We also notice instances where retrieval from the vector database does not contain enough information and the RAG bot generating a non-sensical response and not admitting to its lack of knowledge. In prospect, there is certainly room for optimization such that accuracy and robustness of our RAG bot can be improved. For example, using different text embedding models can make retrieval of documents more accurate alongside carefully splitting the PDF into smaller documents with the right amount of overlap to

preserve semantic meaning over adjacent chunks. In addition, configuring the LLM itself is also a degree of freedom that one can tweak to improve LLM performance and reliability. One can also implement more advanced RAG architectures to question answering accuracy and robustness of the bot[5]. Given the promising evaluation results together with the suggested optimization, we believe that our RAG bot that is more than $10\times$ smaller than chatGPT-3.5, can match the performance of the latter on tasks described in this article.

ACKNOWLEDGMENTS

DATA AVAILABILITY STATEMENT

The code and data that support the findings of this study are openly available on Zenodo with identifier https://doi.org/10.5281/zenodo.10730805

REFERENCES

[1]R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "Hellaswag: Can a machine really finish your sentence?" CoRR **abs/1905.07830** (2019), 1905.07830.

[2]A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," CoRR **abs/1706.03762** (2017), 1706.03762.

[3]S. Lin, A. Wang, and X. Yang, "Sdxl-lightning: Progressive adversarial diffusion distillation," (2024), arXiv:2402.13929 [cs.CV].

[4]Y. Liu, K. Zhang, Y. Li, Z. Yan, C. Gao, R. Chen, Z. Yuan, Y. Huang, H. Sun, J. Gao, L. He, and L. Sun, "Sora: A review on background, technology, limitations, and opportunities of large vision models," (2024), arXiv:2402.17177 [cs.CV].

[5]Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, Q. Guo, M. Wang, and H. Wang, "Retrieval-augmented generation for large language models: A survey," (2024), arXiv:2312.10997 [cs.CL].

[6]Y. Lyu, Z. Li, S. Niu, F. Xiong, B. Tang, W. Wang, H. Wu, H. Liu, T. Xu, E. Chen, Y. Luo, P. Cheng, H. Deng, Z. Wang, and Z. Lu, "Crud-rag: A comprehensive chinese benchmark for retrieval-augmented generation of large language models," (2024), arXiv:2401.17043 [cs.CL].

[7]Y. Ke, L. Jin, K. Elangovan, H. R. Abdullah, N. Liu, A. T. H. Sia, C. R. Soh, J. Y. M. Tung, J. C. L. Ong, and D. S. W. Ting, "Development and testing of retrieval augmented generation in large language models – a case study report," (2024), arXiv:2402.01733 [cs.CL].

[8]R. Yang, T. F. Tan, W. Lu, A. J. Thirunavukarasu, D. S. W. Ting, and N. Liu, "Large language models in health care: Development, applications, and challenges," Health Care Science **2**, 255–263, https://onlinelibrary.wiley.com/doi/pdf/10.1002/hcs2.61.

[9]H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, and A. Mian, "A comprehensive overview of large language models," (2024), arXiv:2307.06435 [cs.CL].

[10]T. Ding, T. Chen, H. Zhu, J. Jiang, Y. Zhong, J. Zhou, G. Wang, Z. Zhu, I. Zharkov, and L. Liang, "The efficiency spectrum of large language models: An algorithmic survey," ArXiv **abs/2312.00678** (2023).

[11]M. Musser, "A cost analysis of generative language models and influence operations," (2023), arXiv:2308.03740 [cs.CY].

[12]J. Bang, Y. Choi, M. Kim, Y. Kim, and M. Rhu, "vtrain: A simulation framework for evaluating cost-effective and compute-optimal large language model training," (2023), arXiv:2312.12391 [cs.LG].

[13]J. Simon, "Large Language Models: A New Moore's Law? — huggingface.co," `https://huggingface.co/blog/large-language-models` (2021), [Accessed 01-03-2024].

[14]C. Nast, "OpenAI's CEO Says the Age of Giant AI Models Is Already Over — wired.com," `https://www.wired.com/story/openai-ceo-sam-altman-the-age-of-giant-ai-models-is-already-over/` (2023), [Accessed 01-03-2024].

[15]E. Almazrouei, H. Alobeidli, A. Alshamsi, A. Cappelli, R. Cojocaru, M. Debbah, Étienne Goffinet, D. Hesslow, J. Launay, Q. Malartic, D. Mazzotta, B. Noune, B. Pannier, and G. Penedo, "The falcon series of open language models," (2023), arXiv:2311.16867 [cs.CL].

[16]H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," (2023), arXiv:2302.13971 [cs.CL].

[17]H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, Y. Li, X. Wang, M. Dehghani, S. Brahma, A. Webson, S. S. Gu, Z. Dai, M. Suzgun, X. Chen, A. Chowdhery, A. Castro-Ros, M. Pellat, K. Robinson, D. Valter, S. Narang, G. Mishra, A. Yu, V. Zhao, Y. Huang, A. Dai, H. Yu, S. Petrov, E. H. Chi, J. Dean, J. Devlin, A. Roberts, D. Zhou, Q. V. Le, and J. Wei, "Scaling instruction-finetuned language models," (2022), arXiv:2210.11416 [cs.LG].

[18]A. Balaguer, V. Benara, R. L. de Freitas Cunha, R. de M. Estevão Filho, T. Hendry, D. Holstein, J. Marsman, N. Mecklenburg, S. Malvar, L. O. Nunes, R. Padilha, M. Sharp, B. Silva, S. Sharma, V. Aski, and R. Chandra, "Rag vs fine-tuning: Pipelines, tradeoffs, and a case study on agriculture," (2024), arXiv:2401.08406 [cs.CL].

[19]"Intel® Core™ i9-9900K Processor (16M Cache, up to 5.00 GHz) Product Specifications — ark.intel.com," `https://ark.intel.com/content/www/us/en/ark/products/186605/intel-core-i9-9900k-processor-16m-cache-up-to-5-00-ghz.html`, [Accessed 28-02-2024].

[20]"Llama — llama.meta.com," `https://llama.meta.com/`, [Accessed 28-02-2024].

[21]B. Vanroy, "Language resources for Dutch large language modelling," arXiv preprint arXiv:2312.12852 (2023).

[22]S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," CoRR **abs/1502.02551** (2015), 1502.02551.

[23]A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," CoRR **abs/2103.13630** (2021), 2103.13630.

[24]onil gova, "K quantization vs perplexity discussion," `https://www.reddit.com/r/LocalLLaMA/comments/1441jnr/k_quantization_vs_perplexity/` (2023), [Accessed 01-03-2024].

[25]V. Rawte, P. Priya, S. M. T. I. Tonmoy, S. M. M. Zaman, A. Sheth, and A. Das, "Exploring the relationship between llm hallucinations and prompt linguistic nuances: Readability, formality, and concreteness," (2023), arXiv:2309.11064 [cs.AI].

[26]H. Chase, "LangChain," (2022).

[27]M. G. Rizzo, N. Cai, and D. Constantinescu, "The performance of chatgpt on orthopaedic in-service training exams: A comparative study of the gpt-3.5 turbo and gpt-4 models in orthopaedic education," Journal of Orthopaedics **50**, 70–75 (2024).

[28]Chromix, "Extensive LLama.cpp benchmark & more speed on CPU, 7b to 30b, Q2_K, to Q6_K and FP16, X3D, DDR-4000 and DDR-6000," `https://www.reddit.com/r/LocalLLaMA/comments/14ilo0t/extensive_llamacpp_benchmark_more_speed_on_cpu_7b/` (2023), [Accessed 01-03-2024].

[29]C. Kachris, "A survey on hardware accelerators for large language models," (2024), arXiv:2401.09890 [cs.AR].

[30]Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, W. Ye, Y. Zhang, Y. Chang, P. S. Yu, Q. Yang, and X. Xie, "A survey on evaluation of large language models," (2023), arXiv:2307.03109 [cs.CL].

[31]Z. Rasool, S. Kurniawan, S. Balugo, S. Barnett, R. Vasa, C. Chesser, B. M. Hampstead, S. Belleville, K. Mouzakis, and A. Bahar-Fuchs, "Evaluating llms on document-based qa: Exact answer selection and numerical extraction using cogtale dataset," (2024), arXiv:2311.07878 [cs.IR].