Development of a chatbot that leverages machine learning

Group 2 PA2595

Thomas Selin
thomasselin.work@gmail.com

Jinni Girhotra
jigi23@student.bth.se

Fredrik Loch
frlo24@student.bth.se

***Abstract*—A modern approach for gaining insights from documents is using large language models (LLM:s) that has been generated by using machine learning.**

We explored three popular methods to enable asking questions about provided documents and receiving meaningful responses. Creating a new LLM, fine-tuning an existing LLM, or using the retrieval-augmented generation method. We discussed each of these methods, focusing on their respective benefits and drawbacks, and compared them in relation the requirements we defined for the chatbot to be able to deliver value to the user.

Subsequently, we designed and implemented a chatbot using the method we deemed most effective, considering the project's scope and other limitations.

During the implementation and testing we focused on improving the quality of the chatbot's answers by other means than changing LLM.

***Index Terms*—Machine learning, RAG, AI, Text generation**

I. Introduction

This project has aimed to develop a chatbot which answers questions about documents provided by the user. We defined the requirements for a useful chatbot of this type and evaluated three different methods to implement this. By comparing each of these methods to our requirements we identified the best method to fulfil our requirements and used it when implementing the chatbot.

A. Project Description

This is a project where we have created a chatbot which answers questions about documents provided by the user. We started with acquiring background knowledge about the subject and then we defined requirements for the chatbot to be of value to the user. We then evaluated different methods to accomplish this. After that was done, we could conclude that only one of the methods was likely to be able fulfil our requirements within the scope of the project. That was the RAG method. We then developed the chatbot software in a modular architecture by using the RAG method.

We tested the software against our requirements. The resulting chatbot fulfilled all requirements in a satisfactory manner.

II. Background

A. Chatbots

Chatbots are software programs crafted to mimic human conversation, engaging with users via text or voice. These applications are widely used across different fields including customer support, information retrieval, and educational settings. The main goal of a chatbot is to deliver prompt and accurate answers to user questions, eliminating the need for human involvement. This efficiency not only cuts down on operational expenses but also enhances service availability and provides uniform answers to frequently asked questions.

The concept of chatbots goes back to the 1960s with the development of ELIZA by Joseph Weizenbaum at MIT. ELIZA was one of the first programs capable of attempting the Turing test, engaging users in a conversation by mimicking therapeutic dialogue [1].The next significant advancement came with the creation of PARRY in 1972, another early chatbot that simulated a person with paranoid schizophrenia [2]. The 1990s saw the development of more sophisticated systems like Dr. Sbaitso and A.L.I.C.E., which featured more advanced pattern matching techniques [3].With the advent of more advanced artificial intelligence and natural language processing techniques, modern chatbots have become increasingly sophisticated. This era includes the introduction of systems like Siri, Alexa, and Google Assistant, which can understand complex voice commands and perform tasks ranging from setting alarms to controlling smart home devices [4].The integration of deep learning techniques has led to significant improvements in chatbot technology. Modern chatbots can learn from vast amounts of conversational data, enabling them to understand context, manage nuanced conversations, and respond in a more human-like manner [5]

As chatbot technology has evolved from simple pattern matching in early versions like ELIZA and PARRY to the more sophisticated interactions of modern assistants such as Siri and Alexa, the core technologies behind them have significantly changed. This shift has been largely driven by improvements in Natural Language Processing (NLP) and Artificial Intelligence. NLP is essential for chatbot development because it enables machines to understand and interpret human language in a meaningful and relevant way. The move to integrate NLP has been fueled by the need for chatbots to manage more complex and diverse conversations, creating interactions that are close to human-like. The following section will explore NLP in detail, specifically how large language models have transformed the way we develop chatbots. These models help chatbots understand, process, and generate human language

with remarkable accuracy and efficiency.

Natural language processing and languages models as a sub-field of generative AI is a relatively new in computer engineering, having been made possible by the advent of large GPU clusters and quick access to massive amounts of data. This technological leap has enabled the creation of sophisticated models that can process and understand human language with unprecedented depth and nuance. The exponential growth in computational power, particularly through GPUs, allows these models to perform complex calculations at high speeds, essential for processing the intricacies of human language. Additionally, the accessibility of massive amounts of textual data from the internet provides the necessary input for training these models, enabling them to learn a wide variety of linguistic patterns and nuances.

Also, when adapting and tuning an available LLM to solve domain specific problems the costs and complexity involved can be particularly prohibiting. Running and updating such a model can incur large costs as well.

The progress in Natural Language Processing (NLP) has paved the way for the development of Large Language Models (LLMs), as a pivotal development in the field of generative AI. This growth has been fueled by technological advances, including large GPU clusters and fast access to huge amounts of data. LLMs are now fundamental to NLP, reaching levels of performance that can rival human skills in specific tasks. These models stand out from earlier NLP technologies because they use much larger datasets and have significantly more detailed modeling parameters. This allows them to understand and use language with a high degree of accuracy and contextual depth.

However, these advancements are not without their significant expenses. The hardware needed to run these large models, along with the extensive time required to train and develop them, makes these systems both resource-heavy and costly. Additionally, the expenses go beyond the initial setup. Adapting and fine-tuning these large language models to meet specific needs in different areas introduces more complexity and further costs. Maintaining and updating these models over time also adds to the ongoing expenses. Moreover, using a large language model in real-world applications requires careful consideration, especially in complex or sensitive areas. Developers face various trade-offs as they try to balance the improved functionality of these models with issues like privacy, data sensitivity, and the complexity of the systems. This section of the report will delve into these challenges and the strategic choices involved in using large language models for chatbot development. It aims to give a clearer insight into what these advanced tools can achieve and their limitations when applied in practical scenarios.

B. Large Language models

Natural language processing and languages models as a sub-field of generative AI is a relatively new in computer engineering, having been made possible by the advent of large GPU clusters and quick access to massive amounts of data. Large language models (LLM:s) have emerged as the defacto standard for natural language processing and generation and have begun to rival human abilities in specialized tasks. LLM's are set apart from previous generations of natural language processors by the significant increase in both modeling parameters and dataset size. This improvement in quality therefore comes with a major cost in the form of hardware capacity and training time making training and developing new or specialized models costly and time consuming.[6]
When adapting and tuning a available LLM to solve domain specific problems the costs and complexity involved can be prohibiting. Running and updating such a model can incur large costs as well. When integrating a LLM into software there are many further limitations and necessary trade-offs to take into consideration, especially when working within complex and highly specific domains or when working with sensitive data.

C. Retrieval augmented generation

In its most basic sense RAG allows a LLM to index and analyze context from databases or documents outside of its initial training. For this basic augmentation the process is split into three stages:

Indexing: Cleans and extracts data from the relevant source, the data is then cleaned up and formatted as plain text. This data is then segmented into chunks that are encoded as vectors using a specific embedding model.

Retrieval: When the model receives a query it performs the same operations as the initial indexing on the query data. The similarity of the encoded vector representation of the query is calculated against the indexed documents using a geometric algorithm. The solution then feeds the generation model the most similar vectors as context to the query.

Generation: The query and context are combined into a prompt to the LLM, which then tries to generate a relevant response.

This simplistic solution has some challenges that mostly stem from the challenge of indexing and retrieval where the shape and content of the initial vectors can have a major impact on the end result. This can become evident when the model lacks part of the required context or becomes overly reliant on the source documents failing to generate a complete response. [7]

III. Software development phases

A. Ideation

We first acquired necessary background knowledge on the concept of LLM:s and different methods of using LLM:s in a software system for our purpose. This knowledge was acquired mainly from the lectures in the course and from various internet resources. We then discussed these different methods considering the scope and limitations in the project.

We found three methods that could possibly be used:

- Creating a new language model
- Fine-tuning an existing language model
- Retrieval augmented generation, which is in described more in the Design and architecture section

These methods were discussed and compared further during the design phase. A combination of either creating a new model and RAG or fine-tuning and RAG could also be an alternative but was excluded due to the limited time scope of the project.

B. Requirements elicitation

We discussed among ourselves which the essential requirements for a chatbot of this type are to be able to answer questions in a way that gives significant value to the user. As we all had experience from earlier of using various chatbots, for example ChatGPT, Gemini and GitHub Co-Pilot, we could use those experiences as background knowledge of what we think a good chatbot is, when defining the requirements..

We decided that the project should have the following requirements:

- Should generate answers by using a LLM trained by machine learning.
- Should be able to answer questions about text in one or more popular document formats.
- Should be a modular system that is made up of containerized modules. Because many applications of ML/AI require large resources in terms of computations and storage, they are usually deployed as a distributed system, using containers. [8]. Hernandez and Miguel also describe that this provides the ability to share resources and that many infrastructures are nowadays migrating to containers for a number of reasons.
- Should present reference to which document was used as additional context together with the response to the user's question. This we think is useful if the user wants to read further on the subject of the question, to verify that the answer is accurate and that the source that was used is something the user accepts as a source. The user might find the source unacceptable for example if it is deemed too old, or of too low quality.

Although chatbots such as ChatGPT can facilitate cost-effective text generation and editing, factually incorrect responses (hallucinations) limit their utility.[9]. Walters and colleagues mentions how this includes fabricated citation/references.

By instead giving direct reference to which document was used as source for answering the question, there will be no fabricated or hallucinated references.

We also set the following additional requirements:

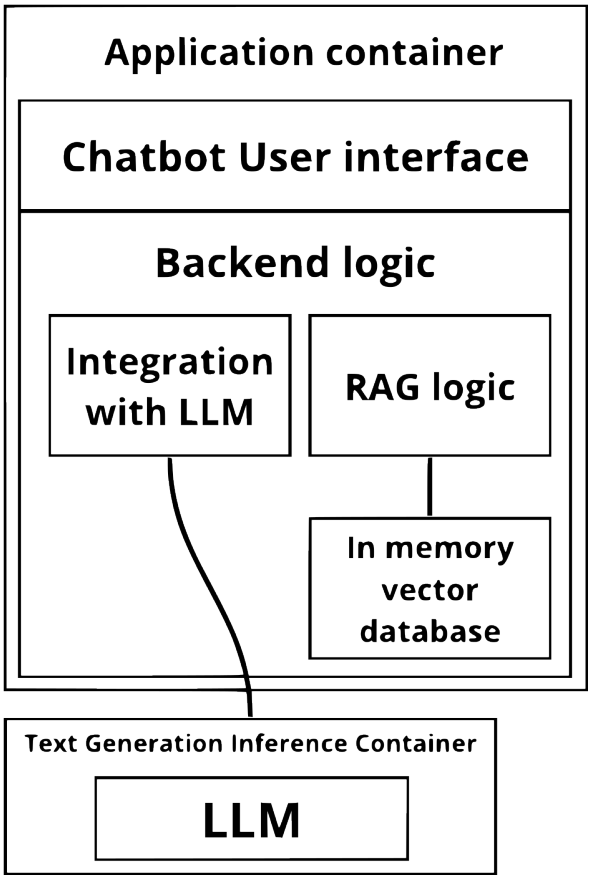
- Should allow specifying which among supported language models to use. This will enable the chatbot to run on a larger variety of computers, servers, or clusters because the choice of language model sets requirements on the hardware resources.
- Should be able to update the documents used as sources without significant complexity and costs.
- Should be implementable within the scope of this course and should be able to be developed, tested, and run on modern personal computers, not using any paid services.

C. Design and architecture

We focused firstly on the language model component as that is central to the system and is the area we had less experience in. We discussed how we best can get a language model to answer questions about document that the user of the chatbot provides. With our acquired background knowledge and defined requirements, we discussed the three different possible methods we identified as candidates in the ideation phase and decided which method to use.

Secondly, we looked for tools and frameworks that would be of use for implementing the chatbot application.

Lastly, we created a design for the overall system to meet our requirements. Here we gave much attention on how to integrate the application with the language model.



1) Language model component: The first method we considered was creating a new language model where the provided documents should be a part of the training data. This approach did not fulfill our requirements as it cannot give reliable references and would most likely need a lot of computational resources which we didn't have access to without cost.

The second method we considered was fine-tuning an existing language model where the provided documents should be a part of the fine-tuning data. This approach did not fulfill our requirements as it cannot give reliable references and would most likely also need a lot of computational resources which we didn't have access to without cost.

The third method we considered was using the retrieval

augmented generation (RAG) method. It involves providing an existing language model with additional context without any further training or tuning of the model. This enables the language model to answer questions about unseen documents that the user provides in the additional context. This approach did fulfill all our requirements and we decided that this was the method that we were going to use. RAG can be easily updated with new documentation without retraining, thus fulfilling the requirement of allowing to use the chatbot on low hardware. Also, because there is no need for training a model, less hardware is required and therefor easier to host on premise instead of in a cloud environment. This makes RAG a good choice for individuals or businesses with limited hardware resources or handles sensitive information that they don't want to be sent over the internet. [7]

We identified a significant value in many real-world scenarios of having the language model served separately from the application as this enables several applications to use the same model server. This led us to use Text Generation Inference (TGI) framework which we identified as an efficient way to implement this.

2) Application: We found that the library LangChain seemed to be especially well suited for this project as it has support for all the features that we needed for our chatbot. LangChain is an extensive library made for the python programming language.

The design is made of two modules that is deployed in two separate containers:

The design is made of two modules:

- A container that uses the TGI framework for serving the language model via a HTTP API.
- A application that at start up processes the documents with help of a tokenizing/embedding model and stores in a in-memory vector database.

The application also provides a chatbot user interface where the user submitts questions. When the user submitts a question the application does the following steps:

1) Performs a similarity search for document that best relates to the question
2) Saves the name of the document which was found most similar
3) Retrieves chunks from that document that are relevant to the question
4) Adds these chunks as additional context
5) Generates the request containing the users question and the additional context
6) Sends requests to the language model API
7) Handles the response which contains the answer to the users question
8) Presents the answer together with reference to the used document

D. Implementation and testing

The chatbot was implemented in a GitHub repository. Along the process of committing code we had continuous discussion among us project members about implementation details. The finished code can be found in the following GitHub repo: Group 2 ML App V1[10] The code fulfills all of our requirements and contains a detailed README file on how to run the chatbot, how to change language model and other essential information.

1) Testing: Testing both under the development process and of the final chatbot has been done in a manual matter via the user interface meanwhile inspecting the application and TGI logs for any errors or warnings. During the testing a few different models was tried:

- bigscience/mt0-small: The smallest tested model with 300 Million parameters.
- stabilityai/stablelm-2-zephyr-1_6b: A model containing 1.6 Billion parameters.
- microsoft/phi-2: The largest tested model with 2.7 Billion parameters.

The result of testing the final version of the chatbot was impressive in the case of the 2 larger models. Especially phi-2 generated impressive natural phrased answers to various submitted questions.

We did not create a test report as the requirements are of true or false character and we decided only to report on any non-fulfilled requirement. Instead, we summarize the testing below:

The result of testing the final version of the chatbot was satisfactory. All our requirements were met when we used one of the two larger LLM:s. Especially microsoft/phi-2 generated impressive natural phrased answers to the various submitted questions.

The smaller LLM (bigscience/mt0-small) often produced answers that were not useful. For example, the answers were in many cases inaccurate, repetitive, badly formatted or too short.

No errors were found in the logs for neither TGI or the application in the later parts of the project.

IV. Maintenance and deployment

A. Maintenance

Maintenance need to take the following factors into consideration:

- Updating of documents
- Updating of TGI framework, the servers operating system and used libraries as newer become available. This ensures security and fixes bug, but can in some cases introduce new bugs. This becomes less necessary if the application is not experiencing problems and if it is not exposed to the internet.
- Updating or changing of model. This can improve chatbot performance if new improved version of a model becomes available or if more hardware resources is available.

B. Deployment

The main concerns when deploying the chatbot are:

- Quality of generated text: To address this concern the preparation for deployment needs to include testing using

different base models to ensure the quality of the answers in the specific use-case. Monitoring the generated answers is beneficial, either by asking the user or manual testing and then decide if attempts to improve the application should be made. It is especially important to evaluate the quality when new documents has been put in use. Alternatively, public available tools or services can be used for more automatic and detailed monitoring.

- Security: This is partly discussed in the plan for maintenance. Also, standard security measures, such as firewall, access control etc. needs to be considered applied when necessary.
- Performance and stability: It is valuable to monitor standard metrics such as error rate, up-time and response time and take appropriate actions if needed to address identified problems.

Depending on the scale of the roll-out it might be necessary to host multiple TGI instances using a load balancer to allow for multiple users, this would also provide a way to gracefully upgrade the model container without creating service disruptions.

V. Conclusion

We conclude that the developed chatbot is successfully implemented and meets the needs for a chatbot of this type to be useful. Together with the maintenance and deployment plans we feel this is a complete software system. One possible improvement could be to use an external vector database instead of an in-memory database. This would provide better fault tolerance and because of the containerized architecture of the chatbot, could enable horizontal scaling by replicating the application container without needing duplication of data in several instances of the in-memory database. This would be particularly important if the document data is very large.

This project has provided insights into the feasibility and challenges that exist when generating answers to questions about documents that the language model is not trained on. The RAG based design used in the project proved to be useful in generating valuable answers, especially when a more capable language model was used.

There is a definitive correlation between the quality and sophistication of the base models used for both tokenizing/embedding and text generation and the quality of the answers.

One additional challenge regarding the tokenization of the documents, this stage is also sensitive to the batch length used in the process. This determines the models understanding of the context supplied, simplistic tokenization and small batches provides a challenge for a smaller model that is unable to combine batches into a coherent reply.

From a practical perspective the choice of utilizing docker containers in the development provides stability and cooperation benefits at the cost of development time, this is due to the additional build step required during development. This cost is rewarded by benefits in the later stages of the project, especially during distribution and maintenance where the controlled and reproducible environment provides major benefits.

During development we have evaluated the performance using both CPU and GPU accelerated models where the main benefit of a GPU accelerated model is generation speed, there is limited or no improvement in quality when using the same models.

When evaluating references, we became aware of the significant difficulty of providing references to the source used when answering the question. This is something we want to highlight for readers as an important thing to consider early when designing a chatbot.

We also want to highlight that the amount of hardware resources available is an important limiting factor when designing a chatbot architecture. If a lot of hardware resources are available, then considering combining RAG with fine-tuning or even creating a new model should be considered.

References

[1] J. Weizenbaum, "Eliza—a computer program for the study of natural language communication between man and machine," *Commun. ACM*, vol. 9, no. 1, pp. 36–45, Jan. 1966, ISSN: 0001-0782. DOI: 10.1145/365153.365168. [Online]. Available: https://doi.org/10.1145/365153.365168.

[2] K. M. Colby, S. Weber, and F. D. Hilf, "Artificial paranoia," *Artificial Intelligence*, vol. 2, no. 1, pp. 1–25, 1971, ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(71)90002-6. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0004370271900026.

[3] R. S. Wallace, "The anatomy of a.l.i.c.e.," in *Parsing the Turing Test: Philosophical and Methodological Issues in the Quest for the Thinking Computer*, R. Epstein, G. Roberts, and G. Beber, Eds. Dordrecht: Springer Netherlands, 2009, pp. 181–210, ISBN: 978-1-4020-6710-5. DOI: 10.1007/978-1-4020-6710-5_13. [Online]. Available: https://doi.org/10.1007/978-1-4020-6710-5_13.

[4] M. B. Hoy, "Alexa, siri, cortana, and more: An introduction to voice assistants," *Medical Reference Services Quarterly*, vol. 37, no. 1, pp. 81–88, 2018, PMID: 29327988. DOI: 10.1080/02763869.2018.1404391. eprint: https://doi.org/10.1080/02763869.2018.1404391. [Online]. Available: https://doi.org/10.1080/02763869.2018.1404391.

[5] Y.-N. (Chen, D. Z. Hakkani-Tür, G. Tür, J. Gao, and L. Deng, "End-to-end memory networks with knowledge carryover for multi-turn spoken language understanding," in *Interspeech*, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:539059.

[6] H. Naveed, A. U. Khan, S. Qiu, *et al.*, *A comprehensive overview of large language models*, 2024. arXiv: 2307.06435 [cs.CL].

[7] Y. Gao, Y. Xiong, X. Gao, *et al.*, *Retrieval-augmented generation for large language models: A survey*, 2024. arXiv: 2312.10997 [cs.CL].

[8] J. A. Hernández and M. Colom, "Repeatability, reproducibility, replicability, reusability (4r) in journals' policies and software/data management in scientific publications: A survey, discussion, and perspectives," *arXiv preprint arXiv:2312.11028*, 2023.

[9] W. H. Walters and E. I. Wilder, "Fabrication and errors in the bibliographic citations generated by chatgpt," *Scientific Reports volume 13, Article number: 14045*, 2023.

[10] F. L. Thomas Selin Jinni Girhotra, *GitHub - Thomas-Selin/BTH-Machine-learning-engineering-project-Group-2 — github.com*, https://github.com/Thomas-Selin/BTH-Machine-learning-engineering-project-Group-2, [Accessed 22-05-2024].