Use of LLM for Methods of Information Retrieval

Jose Gavilanes
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
j.o.gavilanestello@student.utwente.nl

Yancho Bozhilov
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
y.b.bozhilov@student.utwente.nl

Ujjwal Dodeja
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
u.dodeja@student.utwente.nl

Georgios Valtas
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
g.valtas@student.utwente.nl

Alen Badrajan
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
a.badrajan@student.utwente.nl

***Abstract*—This paper focuses on the use of Large Language Models (LLM) in searching, sorting and retrieving information from large chunks of data. In the proposed prototype of the paper, the chunks of data consist of documents such as PDFs and text files. The purpose of such prototype is to show its capabilities on large amounts of data and how it compares to the traditional way of searching which involves manually navigating through folders and files and using your own cognitive methods to summarise an answer to what you are looking for. Just as the traditional way of searching through files, it is a crucial requirement to maintain the same level of security of the data for the LLM method.**

***Index Terms*—LLM, Llama2, LlamaIndex, offline.**

I. Introduction

In the past decades, the amount of data available online has grown exponentially, and necessitated the development of innovative solutions to process vast amounts of data with minimal human effort. In the context, the utilization of Large Language Models (LLMs) have evolved as the most prominent and effective approach.These models comprise of thousands of nodes interconnected into a neural network. With appropriately configuration, they exhibit a form of intelligence by synthesizing knowledge through these connections. As such, the LLMs provide the following advantages:

- Natural Language Comprehension: LLMs can understand easily the text and the way of writing of humans
- Adaptability: LLMs can be fine-tuned into being used for various domains such as virtual assistants to chat bots for entertainment
- Efficiency: After an LLM is trained, its performance is quick and light

With these advantages, LLMs can be seen as very useful when it comes to a task such as the topic of this paper, which is to be used as a local information retrieval tool. However, it is worth noting the disadvantages:

- Bias: The data that the LLM is trained on can be biased, or accidentally the model can learn wrong connections that can lead to misinformation.
- Ethics: As a result of the first disadvantage, the misinformation can lead to fake news or misleading information for ethnic groups and/or genders.
- Environment: Due to the widespread usage of LLMs, the energy required to train and re-train models is enough to affect the environment

This paper focuses on exploring the potential of certain LLMs, and their compatibility during the development of a proposed solution with concrete requirements.

II. Proposed system

The proposed system is a Large Language Model working in a Client-Server architecture that can be hosted on a private network with a centralised database. The goal of the system is to assist an individual in retrieving specific information from a vast amount of data without manual and extensive search. It aims to save the time and effort put into information retrieval while keeping the data secure from public LLMs with the same capability.

III. Requirements

For the aforementioned system mentioned in the previous section, the following requirements were set prior to the developing phase. The requirements are split into two categories: functional and qualitative. Functional requirements set the rules of how the system will behave and how it perform while qualitative requirements set the standards of how well the system is developed to ensure smooth usability.

A. Functional Requirements

- Data Ingestion:
  - Ability to connect to various data sources (Confluence, databases, file systems, GitHub, etc.).
  - Support for multiple data formats (PDFs, DOCX, CSV, etc.).
- Data Indexing:
  - Parsing and indexing of ingested data into a searchable format.

  - Metadata extraction and indexing for advanced search capabilities.
- Search and Retrieval:
  - Natural language processing capabilities for understanding search queries.
  - Fast and efficient search algorithms to handle large datasets.
  - Relevance-based ranking of search results.
  - The system must display the name of each document in search results to allow users to quickly identify the correct file.
- Integration with LLMs:
  - Seamless integration with large language models for enhanced data comprehension.
  - Ability to update and train models with new data.
- User Interface:
  - Intuitive user interface for conducting searches and viewing results.
  - Personalized search options (Retrieve a specific number of sources).
  - Integration with existing company tools such as NVIDIA cards.
- Data Storage and Access:
  - The system must be capable of operating in a completely offline mode, with no requirement for internet connectivity.
  - Data connectors must be designed to work with local network storage solutions or on-premises databases without exposing data to external networks.
- Maintenance and Support:
  - Regular updates and patches for the indexing and search system.
  - Technical support for troubleshooting and resolving issues.

B. *Qualitative Requirements:*

- Performance:
  - High throughput to handle simultaneous search queries.
  - Low latency for returning search results.
  - Scalability to grow with the company's data needs.
- Reliability:
  - High availability with minimal downtime.
  - Fault tolerance and robust error handling.
  - Backup and disaster recovery mechanisms.
- Usability:
  - User-friendly design for ease of use.
  - Comprehensive documentation and user guides.
  - Training materials for end-users.
- Security:
  - The system must ensure that all data remains within the company's private network and is not transmitted over the internet.
- Maintainability:
  - Modular architecture for easy updates and maintenance.
- Compliance:
  - Adherence to industry standards and best practices.
  - Compliance with relevant legal and regulatory requirements.

IV. Planning

This section focuses on how the project was planned in order to have an organized workflow with minimal conflicts, delays, and mistakes. To achieve this, the team needed to effectively manage their time. The expected time period for delivering the system was 10 weeks. With the increasing uncertainty in the specifications of modern day systems, the team chose to work with agile methodology called SCRUM. The team further divided the 10-week period into 5 different sprints consisting of 2 weeks each. The team decided to cover the different phases on software development during these sprints. At the end of each sprint, the team reflected and integrated their evolving understanding into the system design. For a successful outcome, it was deemed necessary to ensure client satisfaction which was achieved by consistent communication with the client for regular updates via email. This helped us with maintaining a clear and common picture of the final product.

A. *Overview*

The following is a Gantt Chart that reflects on how the sprints were planned. Most of the tasks were completed as planned except the tasks from Sprint 4. The tasks from Sprint 4 specifically the GitHub and compatibility debugging, took more time than expected as the team ran in GPU issues which required the team to a lot more resources towards compatibility issues. These issues are thoroughly explained in other sections of this paper. This reduced the amount of time available for testing during the last sprint. Apart from the regular updates, the team conducted two in-person meetings with the Client marked clearly in the Gantt Chart as 'Client Interview' and 'Progress'. The purpose of these meetings were requirement elicitation, and progress update with demo respectively.

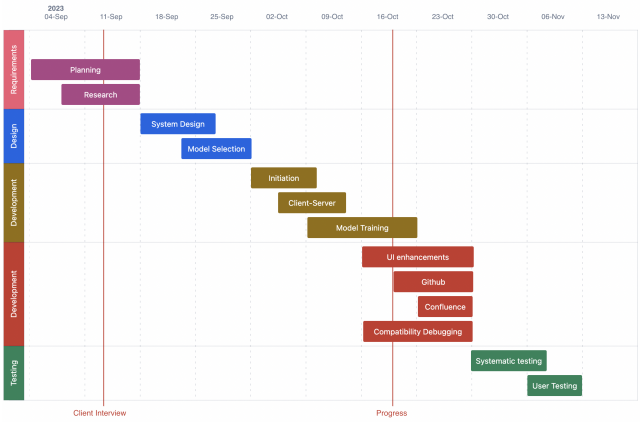

Fig. 1. Llook.ai Planning

B. Tools

Organisational tools - the team used Trello to divide up tasks and keep track of their progress. GitHub was used for version control while Discord for used for regular communication and meetings. Languages and Frameworks - The team chose to code the User Interface in React, and the Server-Client were implemented in Python Flask.

C. Test plan

The testing of a system can be a tricky task due to the presence of bias in shaping the test results and validation of the project's functionality. The team planned to perform systematic testing to ensure the smooth functioning of various features. Additionally, the team wanted to cover user testing with the help of other external unbiased users to determine the relevance of the provided responses.

1) System Testing: During system testing, the team covered the testing of various system features for correctness which mainly included the uploading of documents, addition of git repositories and logging of the chat history. It also included the trials with the features specific to User Interface such as the Dark Mode. The team also ensured that responses from the server were received for all kinds of queries, and how the performance was altered in each case. The 'Testing and Test Analysis' section of this paper provides more detail into the systematic testing that was performed.

2) User Testing: The team planned on validating the system results in terms of user satisfaction by allowing unbiased users to independently use the system and get their insights. However, this was not achieved due to the disruptions in the team's planning, and the time constraints.

V. Risk analysis

An important aspect of working with a client is to ensure the safety of the project that is being developed. The following risk that are listed below have been selected based on how relevant they are to our project and client requirements.

- Data security:
  - Risk: Unauthorized access to sensitive documents.
  - Mitigation: Implement robust encryption for stored documents. Ensure proper access controls and authentication mechanisms.
- Local LLM Security:
  - Risk: Security vulnerabilities in the local LLM.
  - Mitigation: Regularly update the local LLM software, and follow security best practices for local server configurations.
- Privacy of Documents:
  - Risk: Inadvertent exposure of private document content.
  - Mitigation: Apply strong access controls to ensure that only authorized users can query and retrieve documents.
- Query Privacy:
  - Risk: Potential leakage of sensitive information through queries.
  - Mitigation: Implement query anonymization techniques and only store essential query metadata.
- Retrieval Accuracy:
  - Risk: Inaccurate retrieval of documents.
  - Mitigation: Regularly train and fine-tune the local LLM (Llama2) to improve retrieval accuracy. Implement a feedback loop for users to report inaccuracies.
- User Query Understanding:
  - Risk: Users may struggle to formulate effective queries, leading to inaccurate or irrelevant results.
  - Mitigation: Provide clear and concise instructions or examples for formulating queries. Implement a natural language interface that assists users in refining their queries. Offer suggestions or auto-complete features to guide users in constructing well-formed questions.
- Prompt Engineering:
  - Risk: Inaccurate retrieval of documents.
  - Risk: Users might not be familiar with effective "prompt engineering" techniques to extract desired information [4].
  - Mitigation: Provide a user-friendly interface that guides users through constructing effective prompts. Offer templates or pre-defined query structures for common tasks. Implement an educational component within the system, offering tips on constructing optimal queries.
- Continuous Education:
  - Risk: Users might not stay updated on best practices for effective querying.
  - Mitigation: Periodically share tips, updates, and best practices with users. Foster a community where users can share insights and learn from each other. Offer regular webinars or training sessions for advanced query techniques.
- Feedback Mechanism:
  - Risk: Inadequate user feedback leading to unaddressed issues.
  - Mitigation: Implement an easily accessible and user-friendly feedback mechanism. Actively address and resolve user feedback.
- GPU Compatibility Issues:
  - Risk: Compatibility issues with specific GPU models.
  - Mitigation: Choose widely supported GPU models. Regularly update GPU drivers to ensure compatibility with the latest software updates.
- Resource Allocation Issues:
  - Risk: Improper allocation of GPU resources leading to performance degradation.
  - Mitigation: Implement dynamic resource allocation mechanisms. Regularly monitor and adjust resource allocations based on system usage patterns.
- Continuous Improvement:
  - Risk: Lack of continuous improvement in the system.
  - Mitigation: Establish a systematic process for regular

updates and improvements. Stay informed about advancements in local LLM technology.

VI. System Design

After implementing the functional and qualitative requirements, the following design was realised which did not have a lot of differences as compared to the initial system design.

A. Class Diagram

Fig 2 shows the functional components that make up the Llook.ai system. The following classes represent code written in Python and for one class, ChatLog, written in ReactJS. The only exceptions are the LLama7b Model and the Document classes. The Llama7b Model class represents the model as an entity which serves the purpose of processing the documents and the query in order to provide a response. The Document class is a generalization of all the ways to import documents into the model. These documents are then sent to the index server in order to be formatted and organized into chunks of data.

The index_server class is the core component of this system. This class is responsible for sending the response to the API as well as indexing all the documents which are then run through the Llama7b model. The code for running this model is also in the index_server class.

The response given to the flask_api class is then sent to the ChatLog class which is written in ReactJS. Many of the React classes in the user interface were omitted from the class diagram in order to show only the fundamental functionality of the system.

Lastly there are three ways to upload documents to the server, either by pulling all the javadoc from a Github repository, syncing with the documentation of a Confluence account or by uploading a file. In the diagram all these classes are generalized into the Documents class. The Document class, as mentioned before, together with the Llama7b Model class are abstract classes of the diagram in the sense that they are not programming classes with attributes and methods but rather entities that make it more clear in the diagram how the complete system works.

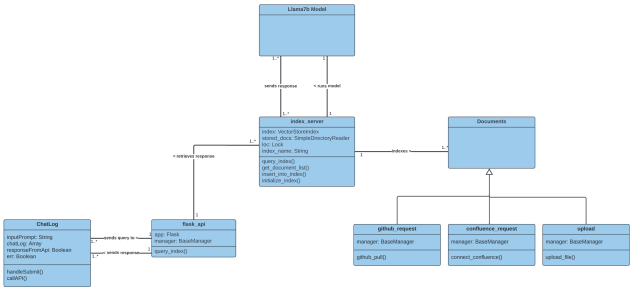

Fig. 2. Class diagram of the functional system of Llook.ai

B. Use Case Diagrams

Fig 3. is a use case diagram demonstrating the processes of the primary user, an employee of the client company, and the features of the system that they are allowed to use. They are able to select the number of sources/documents that they want the response to show. Also for each response, there is the option with a drop-down button to show a larger description of the document text. If the employee needs even more information for their query, they can use the path name of the document and read the document themselves on their company device. This is not part of the diagram as it describes functionality that is beyond the proposed system.The secondary actor of this use case is the system. The functionality of the system is to serve the employee with a response of their query. This functionality is then split into two sub-processes of identifying the location/path of the document and ranking the results in terms of relevance. As the user enters the search prompt, this process includes the system translating the generated embeddings to query which are then compared with the existing data from the database. In such way, the system then identifies the resources' location and calculates the relevance score of indexed paragraphs, providing the most suitable responses. The user can also switch the UI view mode to dark.

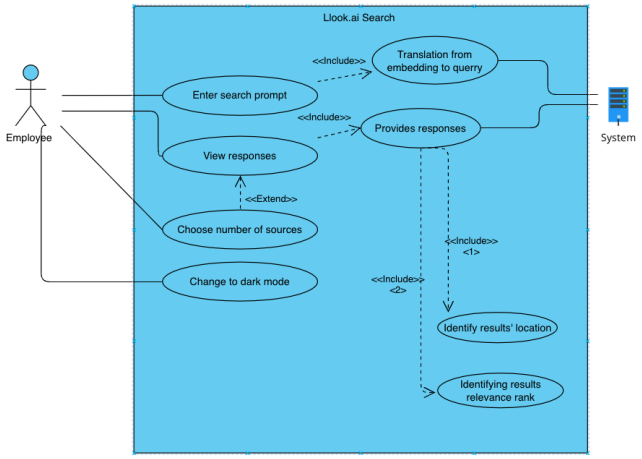

Fig. 3. Llook.ai UseCase Search Diagram

Fig. 4 represents the administration use case of the provided product. The administrator or as the future improvements section would discuss, the CICD process can reiterate the hosting of the system. This case would be used in case of an update of the provided documents to re-index these files. This actor can change the appearance of the system and also upload new documents to the system. The system creates chunks, configures the embeddings [5], and saves the created indexes from the uploaded documents.

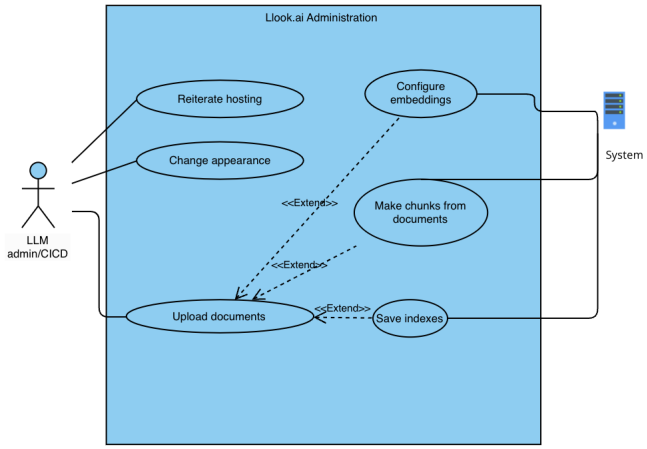

Fig. 4. Llook.ai UseCase Administration Diagram

C. State Machine Diagrams

Figure 5 depicts the state changes at the client side while processing a query given by the user. The 'Initial' state is followed by the 'Idle' state once the system is booted. Once the user has selected the 'Number of Sources' and presses the 'New Chat' button, he enters a new chat room which is presented by a nested state machine 'Chat Room', and the system enters the 'Ready for Input' state. When an input is given by the user. The query is sent to the Server, and a response is awaited which results into two simultaneous states 'Query Sent' and 'Waiting for response'. If a response is received from the server, the response is displayed to the user, and the system gets into 'Response Displayed' state. This is the final state of the 'Chat Room'. –mention the rest If a response is not received from the server, the 'Failure' state is triggered where the system displays an error message. The user can either choose to close the page and get to the 'Final' state or reload the page which reverts back to the 'Ready for Input' state.

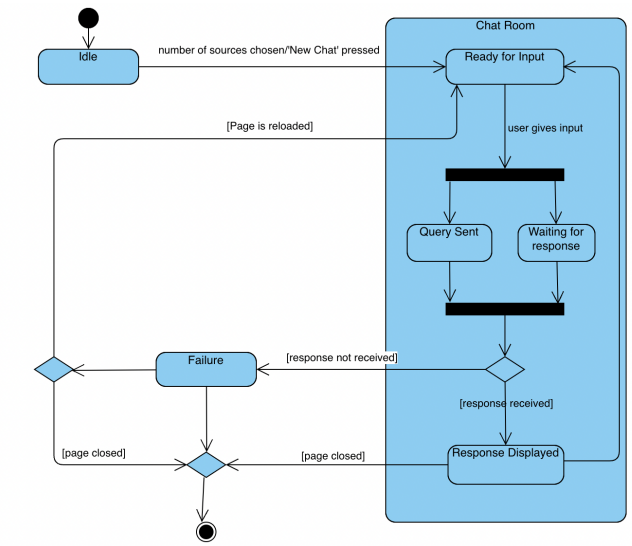

Fig. 5. Llook.ai State Machine Client Querying Diagram

Figure 6 depicts the state changes of the machine while processing a query on the server side received from the client. The system starts in the 'Idle' state. Once a query is received, the system gets into 'Query Received' state and creates an embedding [5] to switch to 'Embedding Created' state. Thereafter, the system looks for similar embedding in the vector database which can results into two different states. If it is found, the system sends the feeds that chunk of information to the LLM, and simultaneously waits for its response. If not found, it gets into a 'Failure' state and forwards the default response. The default response or the LLM response are both written into a JSON putting the machine into 'JSON Created' state. This is followed by the 'Response Sent to Client' state once the response is sent to the client. The 'Final' state comes right after this state.

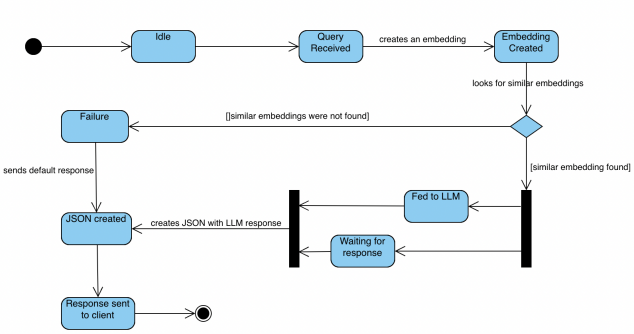

Fig. 6. Llook.ai State Machine Server Querying Diagram

D. Sequence Diagrams

Figure 7 depicts the interaction sequence among four entities, which begins with the collection of 'Documents' that need to be indexed. These documents contain the textual data and associated metadata that will be processed by the LlamaIndex system. LlamaIndex acts as the central processing unit [2]. It ingests the documents and performs initial processing, which includes text extraction, metadata parsing, and any necessary data transformation to prepare the documents for the embedding generation phase. Once the documents are processed, LlamaIndex forwards the textual content to the 'Embedding Generator'. This component utilizes machine learning algorithms to convert the text into high-dimensional vectors, known as embeddings. These embeddings numerically represent the semantic information contained within the text. The embeddings generated by the Embedding Generator are then passed back to LlamaIndex, which in turn stores them in the 'Vector Database'. The Vector Database is optimized for storing high-dimensional vectors and allows for efficient similarity searches.

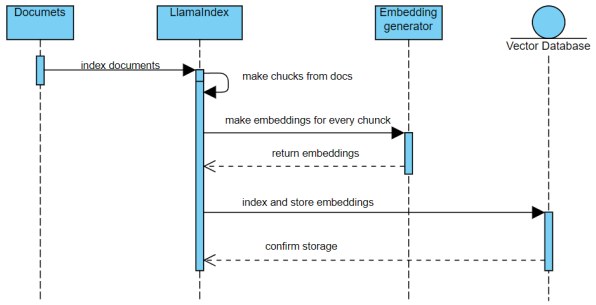

Fig. 7. Llook.ai System Server Diagram

From the client side, the process start as soon as an 'Employee' initiates the process by entering a search query or request for information into the 'UI'. This UI is designed to capture user input in a format that the system can process. The 'UI' sends the Employee's query to the 'Server'. The Server acts as the intermediary, processing the request and orchestrating the subsequent actions required to fulfill the query. Upon receiving the query, the Server performs preprocessing, such as generating a corresponding vector embedding that captures the semantic intent of the query. With this embedding the Server then queries the 'Vector Database' with the embedding to find the most relevant documents or information. The Vector Database uses the embedding to perform a similarity search against its indexed vectors representing documents or data points. The Vector Database returns a list of document identifiers or data points that best match the query embedding which are sent to the LLM. Once the LLM get the information, it tries to contextualize the results or to generate a human-readable summary or response based on the retrieved documents. The Server packages the contextualized information with their corresponding sources and sends them to the UI. The UI presents the search results to the 'Employee', allowing them to review the documents or information retrieved. The results may include links to documents, summaries of content. It is worth noting that the query is being forwarded among all the entities till it reaches the LLM, so that based on that query the LLM can generate an appropriate answer (see Figure 8)

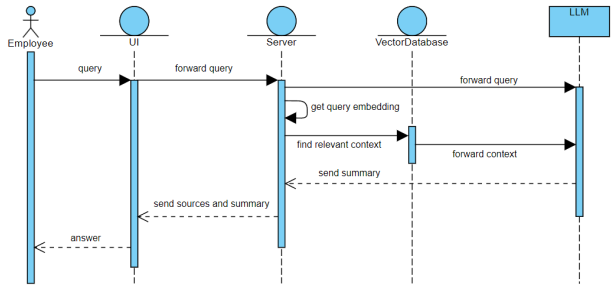

Fig. 8. Llook.ai System Client Diagram

VII. Motivation of Choices

A. Choice of LLM

Llama2 was the team's choice of an LLM. Developed by Meta, LLama2 offers a very comprehensive pre-trained model developed by a reputable company in terms of technological advancements [1]. Furthermore, due to its popularity, it can be assured that new updates will improve it over time as well as not be deprecated due to lack of funding or low user usage. It is provided with a free commercial-to-use license, limited up to 700 million users, which is sufficient for the client, where it has approximately 8K employees around the globe.

Llama2 comes in 3 versions, 7B, 13B and 70B. These three versions refer to the number of parameters that are being used to train the model. For the system produced by the team, 7B proved to be sufficient due to the scope of this project and the outlined requirements by the client. This choice was also influenced by the hardware available to us since all of us could only rely on the available laptop's GPU for testing the system. In the section about hardware requirements, the team go into further detail about what hardware is required for running the different-size models of Llama 2 as well as the results when running models with quantization.

B. Choice of UI

Fig XIII in the appendix refers the user interface of the system. The User Interface was designed with the thought of providing the user with the capability to search for specific information as easily as possible. It was taken into regard that the system should have an intuitive layout which can also be informative when required. The team went through the interfaces of modern day systems who tend to provide similar functionalities. The ChatGPT interface was the biggest source of motivation as it provided the right functionalities as visualised by the team. As a result, the basic layout for the webpage was finalised including the side menu container, search bar prompt and the prompt for the reply by the LLM. With the developing design of the system, the side menu needed a dynamic layout with quite a few options to be added. Initially, the side menu had a 'New Chat' button for the user to start a new chat with the LLM, and all the sources of the response were cited along with the response from the LLM. However, the client requested an option to limit the number of sources to be cited with the response. A 'Select number of sources' drop-down was added in the side menu right below the 'New Chat +' button for the user to be able to do the same. This is followed by a listing of the chat history between the user and LLM, which can be to easily navigate to any of the responses easily. All the modern day systems provide the option to switch to dark mode in order to provide better visibility at night. Therefore, a button for this operation at the top of the side menu with an intuitive icon. The sidemenu also contains the buttons for uploading GitHub repositories or Confluence Links. Once the button is pressed, it asks for the name the link of the repositories.

C. The choice of Administration and Automation

Working with highly sparse and secure data is not an easy task even for a highly trained professional, especially when such information is stored across diverse resources. The client mostly used Atlassian products like Confluence for documentation, BitBucket and Github for repository management, and Jira for ticket handling. As requested by the client: to automate the process of inserting documents into the model, the team decided to automate Confluence and Github, since access to BitBucket API is commercially restricted by Atlassian. Confluence is a website that companies or general people can use it to store, organize and share their documentation. GitHub is a website that allows users and organizations to share, contribute and manage their programming projects. The team's choice for these two services was based on the ability to show the client that they can use the LLM to ask and search normal documents but also code.

1) Github: The client had asked specifically that for code, they prefer only the javadoc or comments and not the actual code to be indexed by the model. Javadoc is way of documenting through annotations what each function does in the code; most commonly used in Java and JavaScript. To use this feature in the app, the user needs to input the Github repository and the creator's username of that repository in order for the app to find it and recursively retrieve all the contents into their respective files for the model to index it. For each file, only the javadoc is extracted and the rest of the code is removed. If a file does not contain javadoc, it remains empty and thus it is not given to the model while the files with javadoc are then converted into PDFs for the model.

2) Confluence: Our client also uses Confluence for organizing and storing the documentation pages. Firstly the team developed a small functionality that takes from the user a Confluence link, and then recursively saves all the pages of that space in pdf files. This script operates on API keys directly from Atlassian Cloud which grants programmatic access to their platform, so in case a future client doesn't host itself Atlassian products, it can still use this integration with Confluence. Llama2 already has the capability to be trained using pdf files, so the team's choice to print the existing webpages to this specific format saves up time for both the development and future administration of this product.

VIII. Hardware Requirements for LLMs

A. GPU vs CPU

Due to the significant improvement of pre-trained language models (LLMs) tackling most natural language processing tasks has become easier than ever. These models have been used in various applications such as chatbots and language translation. When it comes to using LLMs, the choice between using a Graphics Processing Unit instead of a Central Processing Unit might be difficult to understand for some people. In this section, the team will go over the four main reasons why GPUs are still outperforming CPUs when running pre-trained AI models in 2023.

1) Parallel processing: Parallel processing is the execution of instructions simultaneously by using multiple processing units, such as cores of a CPU or GPU. GPUs are designed to perform many operations in parallel, which makes them well-suited for matrix operations which are the fundamental block of most currently used models. This means that a single GPU can perform many calculations simultaneously, greatly reducing the operation time.

2) Memory bandwidth: Using an AI model involves moving large amounts of data between the CPU and the GPU. GPUs have a much higher memory bandwidth in comparison to CPUs, which means that they can move data much faster than the CPUs. This is crucial especially if you would like to train a model since in most cases you will be dealing with massive datasets.

Most recent CPUs have a memory bandwidth of approximately 50-100GB/s, while some of the newer GPUs released by Nvidia go well over 500 GB/s. This discrepancy is achieved by the GPUs having wider memory buses, higher memory clock speeds, and specialized memory architecture.

3) Specialized hardware: As mentioned previously GPUs are designed specifically for performing complex mathematical calculations, while Central Processing Unit (CPU) as the name suggests is designed for more general-purpose computing tasks. This allows the GPU to have more space for processing cores which in return makes them perform calculations much faster. On top of that companies like Nvidia have developed specific cards for AI model training that include a high number of Tensor cores these cores are specifically designed to accelerate the deep learning and AI workloads times. In the later sections, the team will discuss some of the most notable video cards currently available as of November 2023.

4) Cost-effective: Even though GPU prices skyrocketed after the recent chip shortage and crypto boom in 2021-2023 when the team look at the cost-effectiveness of CPUs the team still see a significant amount of upside to choosing a GPU instead. Most of this upside comes in the time needed to complete or run a training task.

B. Nvidia GPUs

To meet the requirements outlined by the client, Thales, it was necessary to ensure the compatibility between the LLM and a Nvidia GPU. This was due to the prevalence of Nvidia GPUs in their onsite servers. In this part of the paper, the team will present a brief overview of the current GPU options, highlighting both their limitations and differences in architectures.

When discussing Graphics Processing Units (GPUs), Nvidia stands out as one of the leaders and pioneers in this space. They are the first company to start optimizing their GPUs to better handle the execution of ML training tasks. As of August 2023, they account for more than 70% [3] of the AI-specialized chip sales.

In this section, the team will examine four different Nvidia GPUs namely A100, V100, T4, and the P100. the team chose these video cards to showcase the different GPU architectures Nvidia has made over the last few years.

1) Pascal: The Pascal architecture was introduced in 2016 and marked a significant improvement in Nvidia's specialized GPU technology. At the time the P100 had an impressive amount of CUDA cores and the use of DDR5 memory which enhanced the parallel processing capabilities of the card. It was built with a 16nm FinFET manufacturing process which improved its power efficiency and overall performance. Even though it was an impressive GPU at the time it is not a suitable option for current LLMs due to the lack of Tensor cores which were introduced in next-generation architecture.

2) Volta: The Volta architecture provided significant improvements for AI training due to the introduction of dedicated Tensor cores. These cores were made as specialized hardware to accelerate the matrix operations which are crucial for deep learning tasks. Along with this, they introduced the second-generation High Bandwidth Memory (HBM2) and NVlink. These technologies introduced very noticeable improvements for multi-GPU configurations. Currently, this architecture is the most cost-efficient on the market if performance is not a strict requirement.

3) Turing: The later introduced Turing architecture builds upon what the Volta architecture introduced. Most notably it has DDR6 compatibility and has increased the number of both CUDA and Tensor cores.

4) Ampere: The latest GPU architecture introduced by Nvidia is Ampere it currently represents the pinnacle of computational power when it comes to AI. Ampere introduced a substantial increase in both CUDA and Tensor core count as well as support for PCI 4.0 which increased the bandwidth for data transfer. Additionally, this architecture supports Nvidia Multi-Instance GPU which allows a single video card to be split into multiple instances with dedicated resources.

C. Llama Chat Models

Llama 2-Chat is a group of fine-tuned Llama 2 models that are optimized for dialogue use cases. These models are specifically designed to generate human-like responses to natural language input, making them suitable for chatbot and conversational AI applications. There are three main Llama 2-Chat model sizes (7B,13B,70B) all of them are trained on 2 trillion tokens and have a context window length of 4096 tokens. In this part, the team will take a closer look at the use cases for each one and this will give more context for why the team chose this specific model.

The main distinction between the three models is the amount of parameters used for each model. The number of parameters in a pre-trained language model determines its capacity to understand and generate complex language patterns. Generally speaking, more parameters lead to better results in performance, generalization across a variety of tasks, and better contextual representation, but they also require a significant amount of computational power. This is why Meta introduced 3 main sizes each one tailored to specific use case requirements.

As the naming suggests 70B [7] model has an impressive 70 billion parameters and stands out as the optimal choice for a system that prioritizes accuracy above everything else. It has remarkable precision in comprehending and crafting responses but requires a minimum of 30GB VRAM to run. Currently, the smallest available version uses 2-bit quantization in super-blocks containing 16 blocks, each block having 16 weights. Block scales and mins are quantized with 4 bits. This ends up effectively using 2.5625 bits per weight. Even with this quantization it still leaves out the majority of the currently available GPUs if the intended system should run on a single card. From the examples given in Table 1, only the 40GB version of the Nvidia A100 and the 32GB version of the Nvidia V100 would be able to fit this model even with quantization.

On the other hand, the 13B [8] model has 13 billion parameters and offers a very compelling alternative if the system has to have a good balance between performance and accuracy. Unfortunately, all of the recommended versions of this model were out of reach for the available hardware because the required VRAM is 10GB. Nevertheless, the team was able to run the smallest versions of this model that uses 2-bit quantization in super-blocks containing 16 blocks, each block having 16 weights. Block scales and mins are quantized with 4 bits. This ends up effectively using 2.5625 bits per weight which means that the team would need 7.93GB of VRAM. However, the results of this model were less than sufficient for of the intended use.

Lastly, the 7B [9] model has 7 billion parameters and is best suited for cost-effective systems where the system hardware might be an issue. This is the model the team chose to use due to its size and performance in comparison to the smallest version of the 13B model mentioned above. The exact version of the model the team used was "llama-2-7b-chat.Q5-K-M.gguf". It uses 5-bit quantization in super-blocks containing 8 blocks, each block having 32 weights. Scales and mins are quantized with 6 bits which results in 5.5 bpw and requires 7.28GB of VRAM.

IX. Testing and Test analysis

Testing pre-trained models can be a very challenging task due to the inherent complexity of these models. One of the primary difficulties comes from the fact that the model has a diverse range of contexts it can understand and is designed to handle, making it nearly impossible to create a standardized evaluation framework. Additionally, another hurdle is the domain gap between the dataset the team used and the dataset of the client intends to use since that could not be provided to us. This section shows the testing the team did to ensure the usability of the prototype system. As mentioned previously this system should not be thought of as a final product but rather a proof of concept for the client. With this in mind, the team divided the testing into two categories Hardware and benchmarks, and Correctness.

1) Hardware Benchmarks: As previously mentioned one of the biggest obstacles in creating this system was the fact that the team did not have the hardware needed to test all of the versions of Llama 2. This is why the team provided the section ”Hardware Requirements for LLMs” to give an overview of what the client might need if they want to turn this prototype into a system they can use. An important thing to mention here is that the correctness of the model does not depend on the hardware it is running on since a pre-trained model should not vary in its responses to a large extent. It only matters that your system is capable of fitting the model. Of course, the running times might differ a lot. Now the team will discuss the findings using the XPS 9520 with Nvidia 3050ti and Mac M1 (8GB integrated graphics)

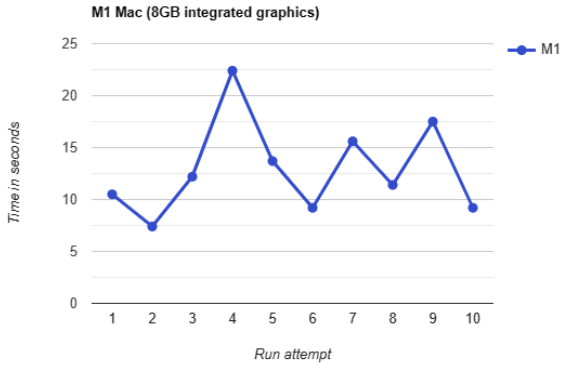

Fig. 9. Time statistics on M1

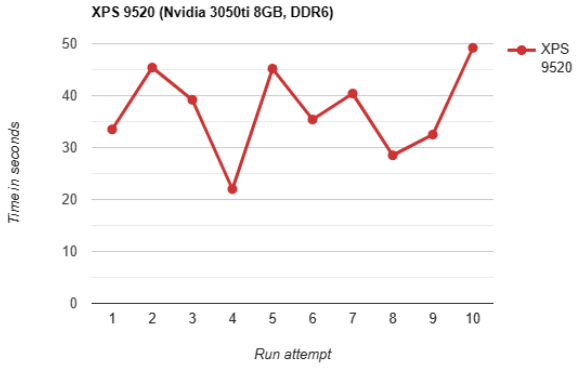

Fig. 10. Time statistics on XPS

The M1 Mac outperforms the XPS by a significant margin as shown in Fig. 9 and Fig. 10. From the findings, this is mainly due to the GPU bottleneck of the system as well as the fact that the memory bandwidth between the CPU and the GPU is higher for the M1. The M1 chip ranges between 7 and 23 seconds with an average of 13.1 seconds while the XPS GPU ranges between 21 and 49 seconds with an average of 29.3 seconds.

The graphs show the time it takes to respond to a user question over 10 trials. For each trial, there was a question asked to both systems in order to observe the GPU performances in different scenarios such as questions regarding documents about the clients technologies to questions about javadoc found in the code.

2) Correctness: To ensure the correctness of the pre-trained model the team conducted testing on diverse datasets, including the provided pdf files from Thales, a self-hosted GitHub repositories with Javadoc, and files for which the team possess a decent level of understanding such as the module 2 guide. Overall the testing methodology involved user-driven testing where the team actively engaged with the model’s responses to test its accuracy and reliability. Using the above-mentioned files the team prepared a series of questions that vary in domain and complexity to test the system. Some of the questions the team included in the final presentation and others will be mentioned below. The team did not think it was necessary to include all of the test cases the team tried due to the fact that the model the team used is from a reputable company and has been tested by others who have documented it online using extensive and sophisticated techniques. Nevertheless here are some key takeaways from the correctness testing.

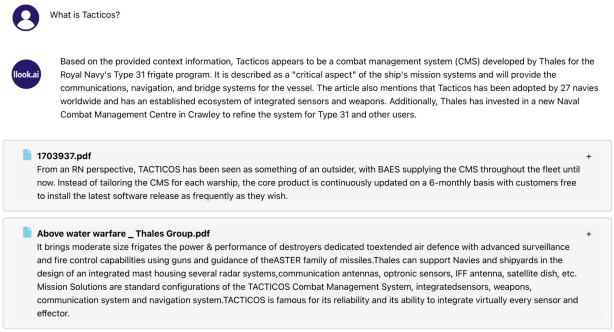

Fig. 11. Question 1

When asking the LLM simple one-dimensional question it performs exceptionally well. As seen in Fig. 11 it can extract key information for a term in the dataset. In Fig.11 the team can also see which are the top 2 indexed files for this term. This is useful for the user since he can navigate to the exact point where the key term is used in every file in the dataset. From the testing, the LLM rarely makes mistakes when asked questions such as in Fig.11

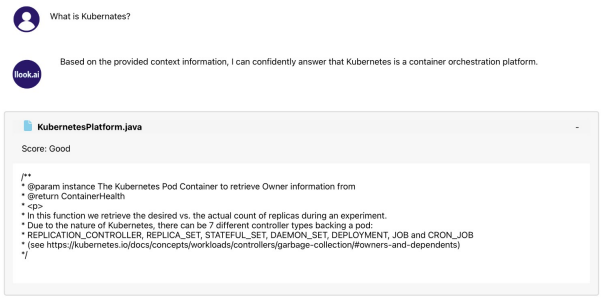

Fig. 12. Question 2

One of the key features we implemented to ensure the correctness of the system for the user is to include a score for how sure we are of the answer from the LLM. For simplicity, we made 3 categories Accurate, Good, and Unreliable this can be seen in Fig. 12. Most of the correct questions we have asked the system result in Good this is by design to not confuse the user with multiple different categories.

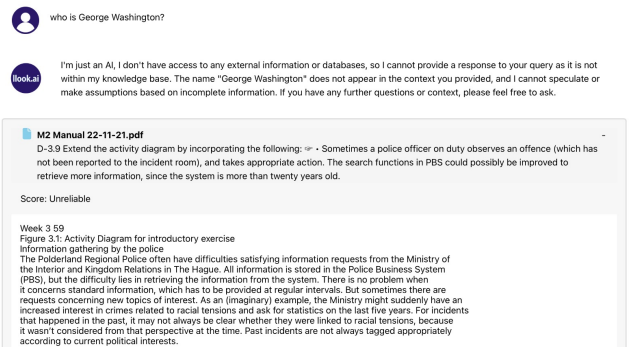

Fig. 13. Question 3

When the system is asked something beyond its scope it will result in a generalized message that informs the user of its capabilities and limitations. One important thing to note is that the system will still try to index and show the user the most relevant information from its dataset but the score will be Unreliable this can be seen in Fig.13 This is because the knowledge base of the LLM does not include any information about George Washington.

X. System Limitations

The final product was able to cover most of the client's requirements, and get a satisfactory response from the Client. However, some of the requirements were partially covered due to limited time-period available for implementation while making sure that minimum functionality was still achieved. This section covers all the partially implemented or ignored requirements during the development of the project.

A. Functional Requirements

1) Data Ingestion: Since Atlassian has a commercial limit to its own BitBucket API, the team had to integrate GitHub as a proof of concept for the integration of Javadoc files.

2) Search and Retrieval: Questioning the system an out-of-the-scope question or asking for non-existing data would provide a standard general response of not being able to properly respond as an LLM system. If the question specifically is about a document indexed by the system, it provides the most relevant paragraphs and their paths.

3) Maintenance and Support: As the client has a strict security policy, the team cannot provide long-term support and troubleshooting for the proposed system unless the team create a new registered business with this concept.

B. Qualitative Requirements

1) Performance: As described in the above Hardware requirements section, the team was quite limited in regards to the available workstations, which led us to pick the suitable version of Llama2 7B due to its indexing time of the documents and fast response to questions close to around 6-12 seconds. The demo machine was based on macOS which had its own integrated GPU with only 8GB VRAM. The available devices while working on this project were limited to only laptops. All of the laptops had GPUs however some were not powerful enough and as a result, the LLM would respond very slowly to the user queries. The much more powerful version of Llama2 70B should satisfy the client because of the client's hardware capabilities.

2) Reliability: We did not implement any disaster recovery mechanisms or monitoring systems since the locally hosted product was one of the main requirements of this project and the client can configure itself the product after the official security standards.

XI. Future Work

The team identified various potential improvements to the system. In the current times, the number of use cases for LLMs is increasing as the technology develops and matures. There are known cases of AI hallucinations as described by [10]. Thus, providing wrong information or generally mixing up the given information. While testing the system on inordinary questions, it has shown the case where an LLM providing no additional information is quite possible. Using appropriate hardware would certainly increase the quality of the responses as well as the provided response time.

The integration with other platforms such as commonly used repositories: GitLab, BitBucket, or other documentation and process management products like Trello or for example Microsoft SharePoint would improve the clearness of the responses due to the availability of diverse sources. Since the provided information was limited to approximately 7 documentation links, the team had to acquire more references to train the model accordingly. The validity and relevance to the client of those files are still under inspection. More data to validate the results would improve the quality of the provided responses. Due to limitations in the user testing, the team was not able to adequately test the user's satisfaction with the provided responses. Acquiring an unbiased user test base to write and perform system user tests would raise the quality and relevance of the responses.

XII. Conclusion

During the span of the project, the team went through different Llama models to understand the potential of Large Language Models with regards to information retrieval from large data sets. It was found that the performance and compatibility have a significant dependence on the hardware specifications of the machine hosting it. The team further developed an understanding of how a large amount of different data sources i.e. PDFs, GIT repositories, Confluence documentation etc., should be processed and indexed in

order to feed the Llama model with specific data, to get efficient responses. In this paper, it was discussed how the team's design choices evolved throughout the project, and how the functionality of the system was able to achieve the right results. It provided an overview of how the hardware specifications and diversely trained LLM models of a machine can effectively change the performance of the final product. Conclusively, it highlighted the limitations of the system by differentiating between the achieved and expected results. In the end, some insights on the potential future improvements were provided.

REFERENCES

[1] Touvron, H. et Al. (2023). Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv preprint arXiv:2307.09288.

[2] Llama-Index Contributors. GPT-Index Documentation. https://gpt-index.readthedocs.io/en/stable/. Accessed: October 10, 2023.

[3] Ross, J. (2023, October 25). Nvidia vs. AMD vs. Intel: Comparing AI Chip Sales. Visual Capitalist. https://www.visualcapitalist.com/nvidia-vs-amd-vs-intel-comparing-ai-chip-sales/

[4] Chen, B., Zhang, Z., Langrene, N., & Zhu, S. (2023). Unleashing the potential of prompt engineering in Large Language Models: a comprehensive review. arXiv preprint arXiv:2310.14735.

[5] Unlu, E. (2023). Structural Embeddings of Tools for Large Language Models. arXiv preprint arXiv:2308.00447.

[6] Reddy, S. (2023). Evaluating large language models for use in healthcare: A framework for translational value assessment. *Informatics in Medicine Unlocked*, 41, 101304. https://doi.org/10.1016/j.imu.2023.101304

[7] TheBloke/Llama-2-70B-Chat-GGUF Â· Hugging Face. (n.d.). https://huggingface.co/TheBloke/Llama-2-70B-Chat-GGUF

[8] TheBloke/Llama-2-13B-chat-GGUF Â· Hugging face. (n.d.-b). https://huggingface.co/TheBloke/Llama-2-13B-chat-GGUF

[9] TheBloke/Llama-2-7B-Chat-GGUF at main. (n.d.). https://huggingface.co/TheBloke/Llama-2-7B-Chat-GGUF/tree/main

[10] Salvagno, M., Taccone, F.S. Gerli, A.G. Artificial intelligence hallucinations. Crit Care 27, 180 (2023). https://doi.org/10.1186/s13054-023-04473-y

XIII. APPENDIX

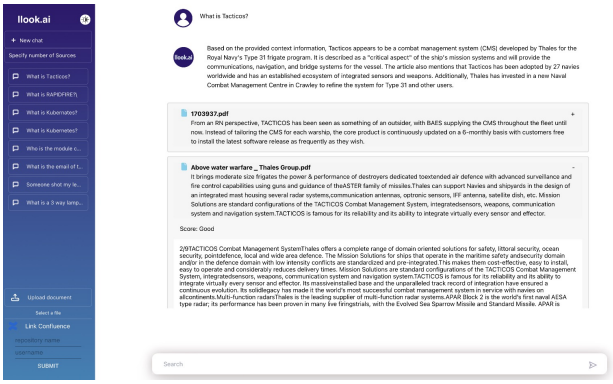

Fig. 14. System's UI

Table I. GPU Specifications

GPU Model	Architecture	VRAM	CUDA Cores	Tensor Cores	Release Date
NVIDIA A100	Ampere	Varied configurations, up to 40 GB	6,912	432	May 2020
NVIDIA V100	Volta	Varied configurations, up to 32 GB	5,120	640	May 2017
NVIDIA T4	Turing	16 GB	16,384	320	September 2018
NVIDIA P100	Pascal	Varied configurations, up to 16 GB	3,584	No (Volta architecture and later introduced Tensor Cores)	April 2016

Table II. Team contributions

Tasks	Contributor
LLM research	Jose and Yancho
Requirements gathering	Everyone
Supervisor/client communication	Ujjwal and Alen
Documents indexing and embeddings	Jose
Github pulling	George
Confluence pulling	Alen and Jose
UI initiation	Jose
Presentation	Yancho and Ujjwal
UI enhancements	Ujjwal
Flask server initialization	Jose
UI and server connection	Ujjwal and Jose
LLM integration	Yancho and Jose
Testing	George, Yancho and Jose
GPU issues	Yancho and Jose
Documentation	Everyone