UNIVERISTY OF PIRAEUS - DEPARTMENT OF INFORMATICS

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ - ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

MSc «AI-Based Model for Knowledge Specific Assistance»

ΠΜΣ «Μοντέλο Τεχνητής Νοημοσύνης για Βοήθεια σε Συγκεκριμένη Γνώση»

MSc Thesis

Μεταπτυχιακή Διατριβή

Thesis Title: Τίτλος Διατριβής:	**AI-Based Model for Knowledge Specific Assistance** Μοντέλο Τεχνητής Νοημοσύνης για Βοήθεια σε Συγκεκριμένη Γνώση
Student's name-surname: Ονοματεπώνυμο φοιτητή:	**Thanos Apostolou** Θάνος Αποστόλου
Father's name: Πατρώνυμο:	**Christos** Χρήστος
Student's ID No: Αριθμός Μητρώου:	**MPSP2203** ΜΠΣΠ2203
Supervisor: Επιβλέπων:	**Dionisios Sotiropoulos, Assistant Professor** Διονύσιος Σωτηρόπουλος, Επίκουρος Καθηγητής

September 2024/ Σεπτέμβριος 2024

3-Member Examination Committee Τριμελής Εξεταστική Επιτροπή		
Dionisios Sotiropoulos **Assistant Professor** Διονύσιος Σωτηρόπουλος Επίκουρος Καθηγητής		

Contents

Abstract

This MSc thesis is about utilizing artificial intelligence models in order to find specific knowledge. As part of this goal we will develop a complete web application, where users will be able to ask questions to artificial intelligence models, which will answer them based on a specific context. We will follow two different methods. For the first method we will create our own text generation AI model [1] which will be trained to understand specific knowledge. For the second method we will use existing artificial intelligence models, trying to limit them so that they respond only to the specific knowledge context that we have chosen. In the end we will be able to come to conclusions about the usefulness of these methods.

Περίληψη

Η παρούσα μεταπτυχιακή εργασία ασχολείται με την αξιοποίηση μοντέλων τεχνητής νοημοσύνης για την υποβοήθηση ανεύρεσης συγκεκριμένης γνώσης. Στα πλαίσια αυτού του στόχου θα αναπτύξουμε μια πλήρη διαδικτυακή εφαρμογή, στην οποία οι χρήστες θα μπορούν να κάνουν ερωτήσεις σε μοντέλα τεχνητής νοημοσύνης, τα οποία θα τους απαντάνε με βάση συγκεκριμένο πλαίσιο. Θα ακολουθήσουμε δύο διαφορετικές μεθοδολογίες. Για την πρώτη μεθοδολογία θα δημιουργήσουμε ένα δικό μας μοντέλο τεχνητής νοημοσύνης παραγωγής κειμένου [1] το οποίο θα εκπαιδευτεί για να κατανοεί συγκεκριμένη γνώση. Για την δεύτερη μεθοδολογία θα χρησιμοποιήσουμε υπάρχοντα μοντέλα τεχνητής νοημοσύνης προσπαθώντας να τα περιορίσουμε ώστε να απαντάνε μόνο στο συγκεκριμένο πλαίσιο γνώσης που έχουμε επιλέξει. Στο τέλος θα μπορέσουμε να καταλήξουμε σε συμπεράσματα. Στο τέλος θα μπορέσουμε να καταλήξουμε σε συμπεράσματα για την χρησιμότητα αυτών των μεθοδολογιών. [2]

1. Introduction

In our era, the knowledge we have acquired is bigger than ever. The number of books, notes, web pages and other forms of content keeps increasing year by year. It is impossible for any human being, to be able to read an process all this available knowledge. Fortunately, technology has been greatly improved and is being used daily for tasks involving knowledge search and analysis. While traditional tools like search engines made it easier for us to find existing knowledge, in the past years we have observed the increasing development of tools using artificial intelligence. We will study the usage of text generation machine learning models in specific knowledge search and analysis assistance. We will use two different methods for these tasks and we will develop a full web application with which users will be able to ask questions

In chapter 2 we will describe and analyze the fundamental theoretical concepts needed for better understanding of this thesis. We will also describe the various technologies and their advantage, which we will use for our application development and deployment.

In chapter 3 we will dive in the details of the two methods that we will use. We will compare them and we will describe their advantages and disadvantages.

In chapter 4 we will describe the architecture and the implementation of our application. We will show the components which construct our application, the tasks each component can perform and how they are connected together.

In chapter 5 we will show the design and execution results of our deployed application. We will investigate the various ways in which our application can be used by the users in order to find specific knowledge based on raw data like documents or web pages.

In chapter 6 we will write our conclusions we reached. We will describe the problems and limitations we faced. Finally, we will specify future improvements that can be made as well as future goals about scaling and expand the core idea.

2. Theory and Literature Review

In this chapter we will talk about the theoretic terms that this thesis is based upon. We will also describe the main technologies which we will use.

2.1. Theoretic Terms

The fundamental theoretic concepts of this thesis stem from the study field of artificial intelligence. We will describe the connection between artificial intelligence, machine learning and deep learning. We will focus on a specific category of machine learning models involving text generation and the state of the art practices of utilizing their capabilities to the maximum by using Retrieval Augmented Generation (RAG) technique.

2.1.1. Artificial intelligence

In the general sense, Artificial intelligence (AI) is intelligence exhibited by machines, particularly computer systems. It is a field of research in computer science that develops and studies methods and software that enable machines to perceive their environment and use learning and intelligence to take actions that maximize their chances of achieving defined goals. Such machines may be called AIs. [3]

Intelligence can be considered to be a property of internal thought processes and reasoning, or a property of intelligent behavior, an external characterization. From these two dimensions (human vs. rational and thought vs. behavior) there are four possible combinations. The methods used are necessarily different: the pursuit of human-like intelligence must be in part an empirical science related to psychology, involving observations and hypotheses about actual human behavior and thought processes; a rationalist approach, on the other hand, involves a combination of mathematics and engineering, and connects to statistics, control theory, and economics. These 4 approaches are the following: [2]

- Acting humanly: The Turing test approach The Turing test, proposed by Alan Turing (1950) and it consists of 4 core principles that a computer would need to follow in order to pass it.
  - natural language processing to communicate successfully in a human language
  - knowledge representation to store what it knows or hears
  - automated reasoning to answer questions and to draw new conclusions
  - machine learning to adapt to new circumstances and to detect and extrapolate patterns

  The full turing test is completed with 2 additional characteristics which have been added by later researchers:
  - computer vision and speech recognition to perceive the world
  - robotics to manipulate objects and move about
- Thinking humanly: The cognitive modeling approach We can determine if a computer or a program thinks like a human by analyzing the human thought in 3 main concepts:
  - introspection - trying to catch our own thoughts as they go by
  - psychological experiments - observing a person in action
  - brain imaging - observing the brain in action
- Thinking rationally: The “laws of thought” approach Rationally thinking can be achieved by following the rules defined by the “logic” study field. When conventional logic requires knowledge that cannot be obtained realistically, then the theory of probability helps us define logical thinking.

- Acting rationally: The rational agent approach Rational thinking can achieve a construction of a comprehensive model of rational thought, but cannot generate intelligent behavior by itself. A rational agent is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome.

2.1.2. Machine Learning

We described the fundamental concepts with which artificial intelligence is defined. Machine learning (ML) is a field of study in artificial intelligence concerned with the development and study of statistical algorithms that can learn from data and generalize to unseen data and thus perform tasks without explicit instructions. [4]

Machine learning is a subset of artificial intelligence (AI) focused on developing algorithms and statistical models that enable computers to perform tasks without explicit instructions. Instead, these systems learn and improve from experience by identifying patterns in data. Machine Learning uses algorithms and statistical models to enable computers to perform specific tasks without being explicitly programmed to do so. Machine learning systems learn from and make decisions based on data. The process involves the following steps:

- Data Collection: Gathering relevant data that the model will learn from.
- Data Preparation: Cleaning and organizing data to make it suitable for training.
- Model Selection: Choosing an appropriate algorithm that fits the problem.
- Training: Using data to train the model, allowing it to learn and identify patterns.
- Evaluation: Assessing the model's performance using different metrics.
- Optimization: Fine-tuning the model to improve its accuracy and efficiency.
- Deployment: Implementing the model in a real-world scenario for practical use.

There are 4 basic types of Machine Learning: [4]–[6]

- Supervised Learning: The model is trained on labeled data, meaning the input comes with the correct output. The goal is to learn a mapping from inputs to outputs. Examples: Regression, classification.
- Unsupervised Learning: The model is trained on unlabeled data, and it must find hidden patterns or intrinsic structures in the input data. Examples: Clustering, association.
- Semi-Supervised Learning: Combines a small amount of labeled data with a large amount of unlabeled data during training. It falls between supervised and unsupervised learning.
- Reinforcement Learning: The model learns by interacting with an environment, receiving rewards or penalties based on its actions, and aims to maximize the cumulative reward. Examples: Game playing, robotic control.

Deep learning is a subset of machine learning that uses multilayered neural networks, called deep neural networks, to simulate the complex decision-making power of the human brain [7]. Deep learning is being used in order to teach computers how to process data in a way that is inspired by the human brain. Deep learning models can recognize complex patterns in pictures, text, sounds, and other data to produce accurate insights and predictions. Deep learning methods can be used in order to automate tasks that typically require human intelligence, such as describing images or transcribing a sound file into text [8]. We can visualize the subsets of Deep Learning, Machine Learning and Artificial Intelligence with the diagram below:

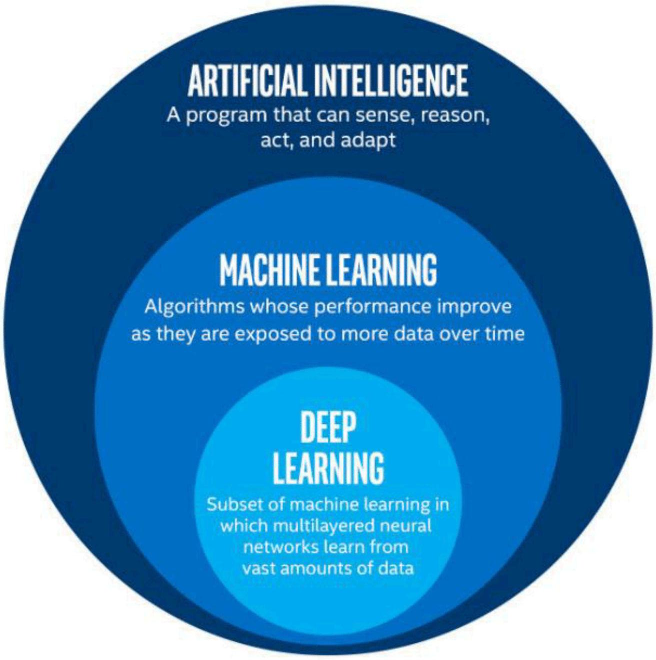
ARTIFICIAL INTELLIGENCE
A program that can sense, reason, act, and adapt
MACHINE LEARNING
Algorithms whose performance improve as they are exposed to more data over time
DEEP LEARNING
Subset of machine learning in which multilayered neural networks learn from vast amounts of data

IMAGE 1: Venn Diagram for AI, ML, Deep Learning [9]

Artificial Intelligence, Machine Learning and Deep Learning are involved in many applications like Image Recognition, Speech Recognition, Traffic prediction, Recommender Systems, Self-driving cars, Email Spam and Malware Filtering, Virtual Personal Assistant, Fraud Detection, Stock Market trading, Medical Diagnosis, Automatic Language Translation, Chatbots, Generation of text images and videos. [10]–[12]. All these applications required different artificial intelligence disciplines that can be combined in order to create a complete artificial intelligence system which produces the required output.

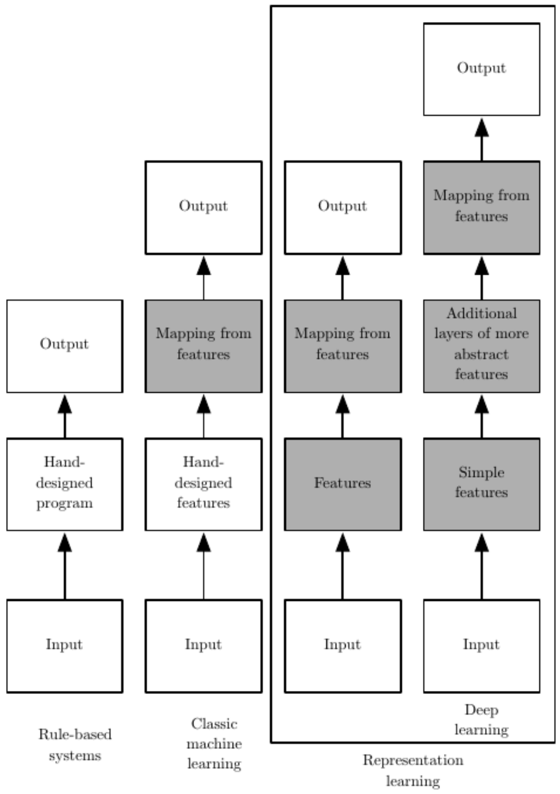

IMAGE 2: Flowcharts showing how the different parts of an AI system relate to each other within different AI disciplines. Shaded boxes indicate components that are able to learn from data. [13]

2.1.3. Text Generation Models, LLM

We described the fundamental concepts of artificial intelligence and machine learning. Now we will take a closer look into a specific category o machine learning models which are used in text generation tasks.

Generative AI refers to deep-learning models that can generate high-quality text, images, and other content based on the data they were trained on. [14]. A text generation model is a type of generative AI models which is designed to produce coherent and contextually relevant textual content. These models are typically based on natural language processing (NLP) techniques and are trained in text data to learn the patterns, grammar, and context required to generate human-like text. When these these models are trained in huge sets of data and have been fed enough examples to be able to recognize and interpret human language or other types of complex data, then they are called large language models (LLM) [15].

These are the key components and concepts of text generation models:

- Training Data:
  - Corpora: Large collections of text used to train the model. These can include books, articles, websites, dialogues, and other text sources.
  - Preprocessing: Cleaning and organizing the text data, including tokenization (breaking text into words or subwords), removing special characters, and normalizing text.
- Model Architecture:

  - Recurrent Neural Networks (RNNs): Earlier models for text generation, including Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRUs), which handle sequential data by maintaining context over time.
  - Transformers: Modern architecture that has become the standard for NLP tasks. Transformers use self-attention mechanisms to process entire sequences of text at once, allowing for better handling of context and dependencies over long distances in the text. Examples include the GPT (Generative Pre-trained Transformer) series, BERT (Bidirectional Encoder Representations from Transformers), and others.
- Training Process:
  - Unsupervised Learning: Most text generation models are trained using unsupervised learning, where the model learns to predict the next word or sequence of words based on the context provided by preceding text.
  - Fine-Tuning: After pre-training on a large corpus, models are often fine-tuned on specific datasets to adapt them to particular tasks or domains.
- Generation Techniques:
  - Sampling: Randomly selecting the next word from the probability distribution generated by the model.
  - Beam Search: An algorithm that searches for the best sequence of words by considering multiple candidate sequences at each step and selecting the most likely ones.
  - Temperature Adjustment: Modifying the probability distribution to control the randomness of the generated text. Lower temperatures result in more deterministic outputs, while higher temperatures produce more diverse and creative text.

  Usually the most popular LLMs have these parameters in order to control sampling. Parameter “top_k” limits the model’s output to the top-k most probable tokens at each step. This can help reduce incoherent or nonsensical output by restricting the model’s vocabulary. Parameter “top_p” filters out tokens whose cumulative probability is less than a specified threshold (p). It allows for more diversity in the output while still avoiding low-probability tokens. Temperature adjusts the randomness or confidence level of the model’s predictions by scaling the log probabilities. Higher temperatures lead to more diverse but potentially nonsensical outputs, while lower temperatures yield more focused and predictable responses [16], [17].
- Evaluation:
  - Perplexity: A measure of how well a probability model predicts a sample. Lower perplexity indicates better performance.
  - Human Evaluation: Assessing the coherence, relevance, and fluency of the generated text through human judges.
  - Automated Metrics: BLEU (Bilingual Evaluation Understudy), ROUGE (Recall-Oriented Understudy for Gisting Evaluation), and other metrics comparing the generated text to reference texts.

Due to the fact that pre-trained LLMs are huge in size and their invocation requires huge amounts of memory, most popular LLMs are converted using some quantization technique. Quantization is a model compression technique that converts the weights and activations within an LLM from a high-precision data representation to a lower-precision data representation, i.e., from a data type that can hold more information to one that holds less. A typical example of this is the conversion of data from a 32-bit floating-point number (FP32) to an 8-bit or 4-bit integer (INT4 or INT8). [18] The are two popular types of LLM Quantization: PTQ and QAT [18]

- Post-Training Quantization (PTQ): this refers to techniques that quantize an LLM after it has already been trained. PTQ is easier to implement than QAT, as it requires less training data and is faster. However, it can also result in reduced model accuracy from lost precision in the value of the weights.
- Quantization-Aware Training (QAT): this refers to methods of fine-tuning on data with quantization in mind. In contrast to PTQ techniques, QAT integrates the weight conversion process, i.e., calibration, range estimation, clipping, rounding, etc., during the training stage. This often results in superior model performance, but is more computationally demanding.

Some advantages of LLM quantization are: [18]

- Smaller Models: by reducing the size of their weights, quantization results in smaller models. This allows them to be deployed in a wider variety of circumstances such as with less powerful hardware; and reduces storage costs.
- Increased Scalability: the lower memory footprint produced by quantized models also makes them more scalable. As quantized models have fewer hardware constraints, organizations can feasibly add to their IT infrastructure to accommodate their use.
- Faster Inference: the lower bit widths used for weights and the resulting lower memory bandwidth requirements allow for more efficient computations.

However the disadvantages are [18]:

- Loss of Accuracy: undoubtedly, the most significant drawback of quantization is a potential loss of accuracy in output. Converting the model’s weights to a lower precision is likely to degrade its performance – and the more “aggressive” the quantization technique, i.e., the lower the bit widths of the converted data type, e.g., 4-bit, 3-bit, etc., the greater the risk of loss of accuracy.

Some thechniques for LLM quantization are the following [18]:

- QLoRA: Low-Rank Adaptation (LoRA) is a Parameter-Efficient Fine-Tuning (PEFT) technique that reduces the memory requirements of further training a base LLM by freezing its weights and fine-tuning a small set of additional weights, called adapters. Quantized Low-Rank Adaptation (QLoRA) takes this a step further by quantizing the original weights within the base LLM to 4-bit: reducing the memory requirements of an LLM to make it feasible to run on a single GPU.
- PRILoRA: Pruned and Rank-Increasing Low-Rank Adaptation (PRILoRA) is a fine-tuning technique recently proposed by researchers that aims to increase LoRA efficiency through the introduction of two additional mechanisms: the linear distribution of ranks and ongoing importance-based A-weight pruning.
- GPTQ: General Pre-Trained Transformer Quantization (GPTQ) is a quantization technique designed to reduce the size of models so they can run on a single GPU. GPTQ works through a form of layer-wise quantization: an approach that quantizes a model a layer at a time, with the aim of discovering the quantized weights that minimize output error (the mean squared error (MSE), i.e., the squared error between the outputs of the original, i.e., full-precision, layer and the quantized layer.)
- GGML/GGUF: GGML (which is said to stand for Georgi Gerganov Machine Learning, after its creator, or GPT-Generated Model Language) is a C-based machine learning library designed for the quantization of Llama models so they can run on a CPU. More specifically, the library allows you to save quantized models in the GGML binary format, which can be executed on a broader range of hardware. GGML quantizes models through a process called the k-quant system, which uses value representations of different bit widths depending on the chosen quant method. First, the model’s weights are divided into blocks of 32: with each block having a scaling factor based on the largest weight value, i.e., the highest gradient magnitude. Depending on the selected

quant-method, the most important weights are quantized to a higher-precision data type, while the rest are assigned to a lower-precision type. For example, the q2_k quant method converts the largest weights to 4-bit integers and the remaining weights to 2-bit. Alternatively, however, the q5_0 and q8_0 quant methods convert all weights to 5-bit and 8-bit integer representations respectively. You can view GGML's full range of quant methods by looking at the model cards in this code repo. GGUF (GPT-Generated Unified Format), meanwhile, is a successor to GGML and is designed to address its limitations, most notably, enabling the quantization of non-Llama models. GGUF is also extensible: allowing for the integration of new features while retaining compatibility with older LLMs.
- AWQ: Conventionally, a model's weights are quantized irrespective of the data they process during inference. In contrast, Activation-Aware Weight Quantization (AWQ) accounts for the activations of the model, i.e., the most significant features of the input data, and how it is distributed during inference. By tailoring the precision of the model's weights to the particular characteristic of the input, you can minimize the loss of accuracy caused by quantization.

2.1.4. Retrieval Augmented Generation (RAG)

Retrieval Augmented Generation (RAG) is a technique for augmenting LLM knowledge with additional data. LLMs can reason about wide-ranging topics, but their knowledge is limited to the public data up to a specific point in time that they were trained on. If you want to build AI applications that can reason about private data or data introduced after a model's cutoff date, you need to augment the knowledge of the model with the specific information it needs. The process of bringing the appropriate information and inserting it into the model prompt is known as Retrieval Augmented Generation (RAG) [19]. RAG extends the already powerful capabilities of LLMs to specific domains or an organization's internal knowledge base, all without the need to retrain the model. It is a cost-effective approach to improving LLM output so it remains relevant, accurate, and useful in various contexts [20].

RAG is important because of these reasons [20]:

- LLMs have known drawbacks:
  - Presenting false information when it does not have the answer.
  - Presenting out-of-date or generic information when the user expects a specific, current response.
  - Creating a response from non-authoritative sources.
  - Creating inaccurate responses due to terminology confusion, wherein different training sources use the same terminology to talk about different things
- RAG comes with additional benefits:
  - Cost-effective implementation: The computational and financial costs of retraining text generation model for organization or domain-specific information are high. RAG is a more cost-effective approach to introducing new data to the LLM.
  - Current information: RAG allows developers to provide the latest research, statistics, or news to the generative models. They can use RAG to connect the LLM directly to live social media feeds, news sites, or other frequently-updated information sources. The LLM can then provide the latest information to the users.
  - Enhanced user trust: RAG allows the LLM to present accurate information with source attribution. The output can include citations or references to sources. Users can also look up source

  documents themselves if they require further clarification or more detail. This can increase trust and confidence in your generative AI solution.

  - More developer control: With RAG, developers can test and improve their chat applications more efficiently. They can control and change the LLM’s information sources to adapt to changing requirements or cross-functional usage. Developers can also restrict sensitive information retrieval to different authorization levels and ensure the LLM generates appropriate responses. In addition, they can also troubleshoot and make fixes if the LLM references incorrect information sources for specific questions. Organizations can implement generative AI technology more confidently for a broader range of applications.

A typical RAG application has two main components [19]:

- Indexing: a pipeline for ingesting data from a source and indexing it. This usually happens offline.
  - Load: First we need to load our data. This is done with Document Loaders.
  - Split: Text splitters break large Documents into smaller chunks. This is useful both for indexing data and for passing it in to a model, since large chunks are harder to search over and won’t fit in a model’s finite context window.
  - Store: We need somewhere to store and index our splits, so that they can later be searched over. This is often done using a VectorStore and Embeddings model.
- Retrieval and generation: the actual RAG chain, which takes the user query at run time and retrieves the relevant data from the index, then passes that to the model.
  - Retrieve: Given a user input, relevant splits are retrieved from storage using a Retriever.
  - Generate: A ChatModel / LLM produces an answer using a prompt that includes the question and the retrieved data

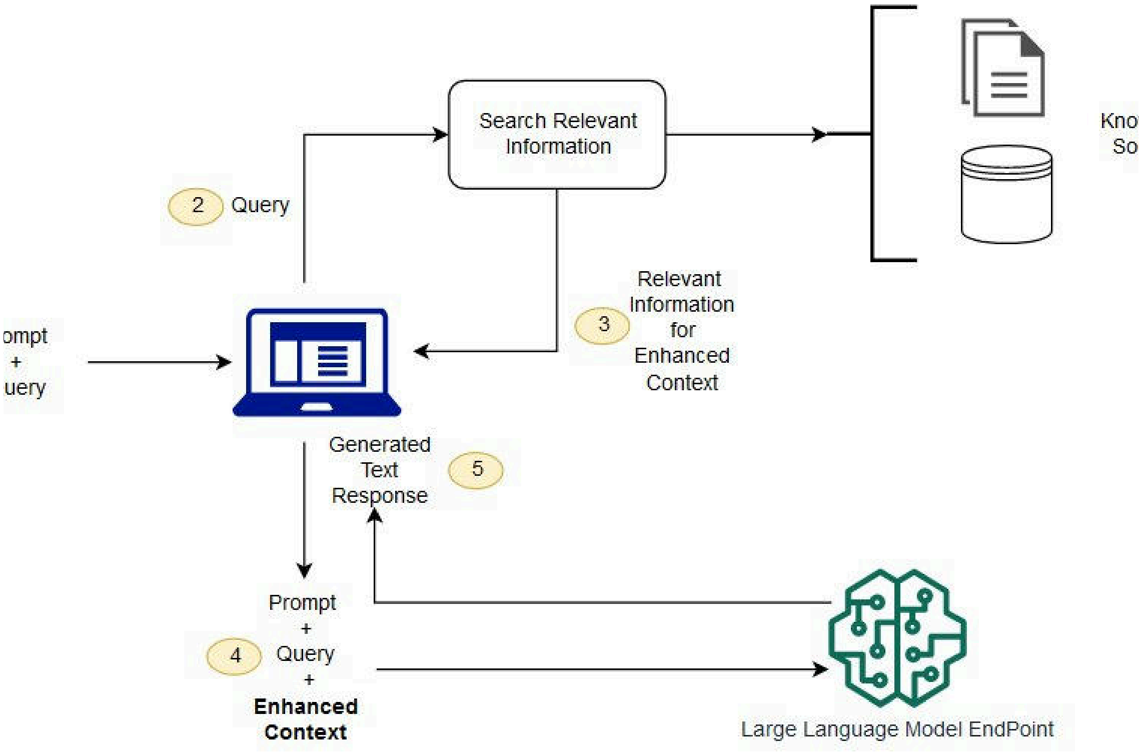
Search Relevant Information
Kno
So
2 Query
ompt
+
uery
3 Relevant Information for Enhanced Context
Generated Text Response 5
Prompt
+
4 Query
+
Enhanced Context
Large Language Model EndPoint

IMAGE 3: Conceptual flow of using RAG with LLMs. [20]

2.2. Technologies

In the context of this thesis we will use many technologies in order to produce a complete application. We will describe the main programming languages we used for this application, Rust and Python. We will see more details about the core programming libraries our application is using and their basic features we utilize. Finally we will talk about the state of the art deployment procedure of deployments based on containers utilization.

2.2.1. Python Programming Language

Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a “batteries included” language due to its comprehensive standard library. [21]

Choosing python as a programming language has several benefits [22]

- Developers can easily read and understand a Python program because it has basic, English-like syntax.
- Python makes developers more productive because they can write a Python program using fewer lines of code compared to many other languages.
- Python has a large standard library that contains reusable codes for almost any task. As a result, developers do not have to write code from scratch.
- Developers can easily use Python with other popular programming languages such as Java, C, and C++.
- The active Python community includes millions of supportive developers around the globe. If you face an issue, you can get quick support from the community.
- Plenty of helpful resources are available on the internet if you want to learn Python. For example, you can easily find videos, tutorials, documentation, and developer guides.
- Python is portable across different computer operating systems such as Windows, macOS, Linux, and Unix.

The python programming language is very popular in various applications: [22]

- Server-side web development: Server-side web development includes the complex backend functions that websites perform to display information to the user. For example, websites must interact with databases, talk to other websites, and protect data when sending it over the network.
- Graphical User Interfaces development (GUI): Develop user friendly GUI applications while also providing a good user experience.
- Gaming development: Python can be used to develop games. Popular python libraries exist for both 2D and 3D graphics development.
- Automation with Python scripts: A scripting language is a programming language that automates tasks that humans normally perform. Such tasks usually include:
  - filesystem operations like files renaming, reading, writing, converting.
  - Mathematical operations
  - download content
  - text operations and transformations in files
  - basic log analysis

- Data science and machine learning: Data science is extracting valuable knowledge from data, and machine learning (ML) teaches computers to automatically learn from the data and make accurate predictions.
- Software deployments and operations: Python can be used for different development tasks and software applications such as:
  - Automatically building the software
  - Utilize continuous integration and continuous deployments (CI/CD)
  - Handling software project management
- Software test automation: Check whether the actual results from the software match the expected results to ensure that the software is error-free.

Using python programming language has various advantages [23]:

- Presence of third-party modules: Python has a rich ecosystem of third-party modules and libraries that extend its functionality for various tasks.
- Extensive support libraries: Python boasts extensive support libraries like NumPy for numerical calculations and Pandas for data analytics, making it suitable for scientific and data-related applications.
- Open source and large active community base: Python is open source, and it has a large and active community that contributes to its development and provides support.
- Versatile, easy to read, learn, and write: Python is known for its simplicity and readability, making it an excellent choice for both beginners and experienced programmers.
- User-friendly data structures: Python offers intuitive and easy-to-use data structures, simplifying data manipulation and management.
- High-level language: Python is a high-level language that abstracts low-level details, making it more user-friendly.
- Dynamically typed language: Python is dynamically typed, meaning you don’t need to declare data types explicitly, making it flexible but still reliable.
- Object-Oriented and Procedural programming language: Python supports both object-oriented and procedural programming, providing versatility in coding styles.
- Portable and interactive: Python is portable across operating systems and interactive, allowing real-time code execution and testing.
- Ideal for prototypes: Python’s concise syntax allows developers to prototype applications quickly with less code.
- Highly efficient: Python’s clean design provides enhanced process control, and it has excellent text processing capabilities, making it efficient for various applications.
- Internet of Things (IoT) opportunities: Python is used in IoT applications due to its simplicity and versatility.
- Interpreted language: Python is interpreted, which allows for easier debugging and code development.

However python programming language has also some drawbacks [23]:

- Performance: Python is an interpreted language, which means that it can be slower than compiled languages like C or Java. This can be an issue for performance-intensive tasks.
- Global Interpreter Lock: The Global Interpreter Lock (GIL) is a mechanism in Python that prevents multiple threads from executing Python code at once. This can limit the parallelism and concurrency of some applications.

- Memory consumption: Python can consume a lot of memory, especially when working with large datasets or running complex algorithms.
- Dynamically typed: Python is a dynamically typed language, which means that the types of variables can change at runtime. This can make it more difficult to catch errors and can lead to bugs.
- Packaging and versioning: Python has a large number of packages and libraries, which can sometimes lead to versioning issues and package conflicts.
- Lack of strictness: Python's flexibility can sometimes be a double-edged sword. While it can be great for rapid development and prototyping, it can also lead to code that is difficult to read and maintain.

2.2.2. Rust Programming Language

Rust is a general-purpose programming language emphasizing performance, type safety, and concurrency. It enforces memory safety, meaning that all references point to valid memory, without a garbage collector. To simultaneously enforce memory safety and prevent data races, its "borrow checker" tracks the object lifetime of all references in a program during compiling. Rust was influenced by ideas from functional programming, including immutability, higher-order functions, and algebraic data types. It is popular for systems programming.Rust does not enforce a programming paradigm, but supports object-oriented programming via structs, enums, traits, and methods, and supports functional programming via immutability, pure functions, higher order functions, and pattern matching. [24]

Rust programming languages can be used in many applications [25], [26]:

- Server-side web development: Rust can be used to write simple REST apis or even full stack backend applications which connect to databases and serve complex html pages using various template engines.
- Client-side web development: Rust can be compile to WebAssembly and be used to create client side web applications supported in all modern web browsers.
- Graphical User Interfaces development (GUI): Develop user friendly GUI applications while also providing a good user experience. Rust gui frameworks can use different architecture including Elm, immediate mode, reactive and others.
- Gaming development: Rust can be used to develop games. New modern rust libraries have emerged for both 2D and 3D graphics development.
- Embedded development: Due to low resource usage and high performance, rust can be used to develop applications for low resource devices.
- Command line development: Rust can be used to create both command line interfaces (CLI) as well as terminal user interfaces (TUI).
- Internet of Things development (IoT): IoT devices typically have limited resources, and Rust's memory safety and low-level control make it an excellent choice for developing embedded systems.
- Robotics: Robotics requires real-time processing, and Rust's low-level control and memory safety make it ideal for developing real-time applications. Rust's concurrency features make it possible to handle multiple threads efficiently, which is essential in robotics applications.
- Machine Learning: Rust libraries and ecosystem for machine learning development is newer and currently more limited than Python's. However rust can be preferred for its higher performance, the memory safety and its immutability features.

Using rust programming language has various advantages [27]:

- Memory Safety: Rust's borrow checker ensures memory safety without the overhead of garbage collection. This means fewer memory leaks and crashes.
- Performance: Comparable to C and C++, Rust provides fine-grained control of memory and other resources.
- Concurrency: Rust's ownership model makes concurrent programming more manageable and less prone to bugs.
- Modern Tooling: Cargo, Rust's package manager and build system, is highly praised for its ease of use.
- Vibrant Community: Rust has a growing and enthusiastic community, which results in good documentation, community support, and an expanding ecosystem.

However rust programming language has also some drawbacks [27]:

- Steep Learning Curve: Rust's unique features, like ownership and lifetimes, can be challenging to grasp for newcomers.
- Compilation Time: Rust programs can have longer compile times compared to some other languages.
- Lesser Library Support: While growing, the Rust ecosystem is still smaller than those of older languages like C++, Java or Python.

2.2.3. Python Libraries

We will talk about the most important Python libraries that we use for our application and their important parts that we utilize.

One of the most significant libraries we use is NLTK. NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum. Thanks to a hands-on guide introducing programming fundamentals alongside topics in computational linguistics, plus comprehensive API documentation, NLTK is suitable for linguists, engineers, students, educators, researchers, and industry users alike. NLTK is available for Windows, Mac OS X, and Linux. Best of all, NLTK is a free, open source, community-driven project [28]. NLTK comes with many corpora, toy grammars, trained models, etc which are called NTLK Data [29]. One of the most important modules of NLTK is the “tokenize” module which can help us split a long text into sentences or into words.

In order to create our own machine learning model we will use PyTorch library. PyTorch is a machine learning library based on the Torch library, used for applications such as computer vision and natural language processing [30]. Written in Python, it's relatively easy for most machine learning developers to learn and use. PyTorch is distinctive for its excellent support for GPUs and its use of reverse-mode auto-differentiation, which enables computation graphs to be modified on the fly. The most fundamental concepts of pytorch are Tensors and Graphs. Tensors are a core PyTorch data type, similar to a multidimensional array, used to store and manipulate the inputs and outputs of a model, as well as the model's parameters. Tensors are similar to NumPy's ndarrays, except that tensors can run on GPUs to accelerate computing. Graphs are data structures consisting of connected nodes (called vertices) and edges. Every modern framework for deep learning is based on the concept of graphs, where Neural Networks are represented as a graph structure of computations. PyTorch keeps a record of tensors and executed operations in a directed acyclic graph (DAG) consisting of Function objects. In this DAG, leaves are the input tensors, roots are the output tensors. [31]

In order to use existing pre-trained Large Language Models (LLM), which we described in Section 2.1.3, we will use LangChain. LangChain is a framework for developing applications powered by large language models (LLMs). LangChain simplifies every stage of the LLM application lifecycle [32]:

- Development: Build your applications using LangChain's open-source building blocks, components, and third-party integrations. Use LangGraph to build stateful agents with first-class streaming and human-in-the-loop support.
- Productionization: Use LangSmith to inspect, monitor and evaluate your chains, so that you can continuously optimize and deploy with confidence.
- Deployment: Turn your LangGraph applications into production-ready APIs and Assistants with LangGraph Cloud.

LangChain does not serve its own LLMs, but rather provides a standard interface for interacting with many different LLMs. To be specific, this interface is one that takes as input a string and returns a string. There are lots of LLM providers (OpenAI, Cohere, Hugging Face, Llama.cpp, etc), the LLM class is designed to provide a standard interface for all of them. [33] LangChain provides also integration with vector stores. One of the most common ways to store and search over unstructured data is to embed it and store the resulting embedding vectors, and then at query time to embed the unstructured query and retrieve the embedding vectors that are 'most similar' to the embedded query. A vector store takes care of storing embedded data and performing vector search for you. Supported vector stores are [34]:

- Chroma
- Pinecone
- FAISS
- Lance

We will use Hugging Face Hub in order to be able to download pre-trained LLMs. The Hugging Face Hub is a platform with over 350k models, 75k datasets, and 150k demo apps (Spaces), all open source and publicly available, in an online platform where people can easily collaborate and build ML together. The Hub works as a central place where anyone can explore, experiment, collaborate, and build technology with Machine Learning [35]

From the supported LangChaing LLM providers we will mostly use the Llama.cpp provider. The main goal of llama.cpp is to enable LLM inference with minimal setup and state-of-the-art performance on a wide variety of hardware - locally and in the cloud. Some characteristics of this library are the following [36]

- Plain C/C++ implementation without any dependencies
- Apple silicon is a first-class citizen - optimized via ARM NEON, Accelerate and Metal frameworks
- AVX, AVX2 and AVX512 support for x86 architectures 1.5-bit, 2-bit, 3-bit, 4-bit, 5-bit, 6-bit, and 8-bit integer quantization for faster inference and reduced
- memory use Custom CUDA kernels for running LLMs on NVIDIA GPUs (support for AMD GPUs via HIP) Vulkan and SYCL backend support CPU+GPU hybrid inference to partially accelerate models larger than the total VRAM capacity

LLama.cpp uses ggml, a tensor library for machine learning [37]. So llama cpp can be used with any LLMs which have been converted to ggml or gguf quantized formats, which we discussed in Section 2.1.3.

In order to create a graph of our own machine learning model performance we will use matplotlib. Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, Python/IPython shells, web application servers, and various graphical user interface toolkits. [38]

We will use unstructured library in order to be able to read various documents. The unstructured library provides open-source components for ingesting and pre-processing images and text documents, such as PDFs, HTML, Word docs, and many more. The use cases of unstructured revolve around streamlining and optimizing the data processing workflow for LLMs. unstructured modular functions and connectors form a cohesive system that simplifies data ingestion and preprocessing, making it adaptable to different platforms and efficient in transforming unstructured data into structured outputs. [39]

We will depend on faiss library to provide us a vector store implementation and search algorithm to use with LangChain. Faiss is a library for efficient similarity search and clustering of dense vectors. It contains algorithms that search in sets of vectors of any size, up to ones that possibly do not fit in RAM. It also contains supporting code for evaluation and parameter tuning. Faiss is written in C++ with complete wrappers for Python/numpy. Some of the most useful algorithms are implemented on the GPU. [40]

2.2.4. Rust Libraries

We will talk about the most important Rust libraries that we use for our application and their important parts that we utilize.

In order to create a command line application (cli) we will use clap library. Clap is a command line argument parser for Rust. I allows us to create our command-line parser, with all of the bells and whistles, declaratively or procedurally. [41]

To use aynschronous programming in Rust we need a asynchronous runtime, so we will use the most popular one tokio. Tokio is an event-driven, non-blocking I/O platform for writing asynchronous applications with the Rust programming language. At a high level, it provides a few major components: [42]

- A multithreaded, work-stealing based task scheduler.
- A reactor backed by the operating system's event queue (epoll, kqueue, IOCP, etc…).
- Asynchronous TCP and UDP sockets.

Tokio provides a runtime for writing reliable, asynchronous, and slim applications with the Rust programming language. It is: [42]

- Fast: Tokio's zero-cost abstractions give you bare-metal performance.
- Reliable: Tokio leverages Rust's ownership, type system, and concurrency model to reduce bugs and ensure thread safety.
- Scalable: Tokio has a minimal footprint, and handles backpressure and cancellation naturally.

We will use axum library to develop our web server. Axum is a web application framework that focuses on ergonomics and modularity. Some high level features are: [43]

- Route requests to handlers with a macro free API.

- Declaratively parse requests using extractors.
- Simple and predictable error handling model.
- Generate responses with minimal boilerplate.
- Take full advantage of the tower and tower-http ecosystem of middleware, services, and utilities.

We will use SeaORM library so that our backend fetches and writes data in a database. SeaORM is a relational ORM which helps us build web services in Rust with the familiarity of dynamic languages. Some SeaORM features are: [44]

- Async: Relying on SQLx, SeaORM is a new library with async support from day 1.
- Dynamic: Built upon SeaQuery, SeaORM allows you to build complex dynamic queries.
- Testable: Use mock connections and/or SQLite to write tests for your application logic.
- Service Oriented: Quickly build services that join, filter, sort and paginate data in REST, GraphQL and gRPC APIs.

SeaORM supports connections with MySQL, Postgres or SQLite databases [45].

2.2.5. HTML, CSS, SPA, Typescript, Vue.js

We will use Web Technologies in order to create a Frontend for our application. In this section we will describe the main technologies for our frontend.

HTML (HyperText Markup Language) is the most basic building block of the Web. It defines the meaning and structure of web content. Other technologies besides HTML are generally used to describe a web page's appearance/presentation (CSS) or functionality/behavior (JavaScript). "Hypertext" refers to links that connect web pages to one another, either within a single website or between websites. Links are a fundamental aspect of the Web. By uploading content to the Internet and linking it to pages created by other people, you become an active participant in the World Wide Web. HTML uses "markup" to annotate text, images, and other content for display in a Web browser. HTML markup includes special "elements" such as `<head>, <title>, <body>` and many others. An HTML element is set off from other text in a document by "tags", which consist of the element name surrounded by "<" and ">". The name of an element inside a tag is case-insensitive. That is, it can be written in uppercase, lowercase, or a mixture. [46]

Cascading Style Sheets (CSS) is a stylesheet language used to describe the presentation of a document written in HTML or XML (including XML dialects such as SVG, MathML or XHTML). CSS describes how elements should be rendered on screen, on paper, in speech, or on other media. CSS is among the core languages of the open web and is standardized across Web browsers according to W3C specifications. Previously, the development of various parts of CSS specification was done synchronously, which allowed the versioning of the latest recommendations. You might have heard about CSS1, CSS2.1, or even CSS3. There will never be a CSS3 or a CSS4; rather, everything is now just "CSS" with individual CSS modules having version numbers. [47]

JavaScript (JS) is a lightweight interpreted (or just-in-time compiled) programming language with first-class functions. While it is most well-known as the scripting language for Web pages, many non-browser environments also use it, such as Node.js, Apache CouchDB and Adobe Acrobat. JavaScript is a prototype-based, multi-paradigm, single-threaded, dynamic language, supporting object-oriented, imperative, and declarative (e.g. functional programming) styles. JavaScript's dynamic capabilities include runtime object construction, variable parameter lists, function variables, dynamic script creation (via eval), object introspection (via for…in and Object utilities), and source-code recovery (JavaScript functions store their source text and can be retrieved through toString()). The standards for JavaScript are the ECMAScript Language Specification (ECMA-262) and the ECMAScript Internationalization API specification (ECMA-402). [48]

TypeScript is a programming language that adds static type checking to JavaScript. TypeScript is a superset of JavaScript, meaning that everything available in JavaScript is also available in TypeScript, and that every JavaScript program is a syntactically legal TypeScript program. Also, the runtime behavior of TypeScript and JavaScript is identical. However, TypeScript adds compile time type checking, implementing rules about how different types can be used and combined. This catches a wide variety of programming errors that in JavaScript are only encountered at runtime. Some typing rules are inferred from JavaScript. ypeScript also enables the programmer to annotate their code, to indicate, for example, the types of parameters to a function or the properties of an object. After compilation, type annotations are removed, making the compiled output just JavaScript, meaning it can be executed in any JavaScript runtime. [49]

An SPA (Single-page application) is a web app implementation that loads only a single web document, and then updates the body content of that single document via JavaScript APIs such as Fetch when different content is to be shown. This therefore allows users to use websites without loading whole new pages from the server, which can result in performance gains and a more dynamic experience, with some tradeoff disadvantages such as SEO, more effort required to maintain state, implement navigation, and do meaningful performance monitoring. [50]

Our frontend will utilize all the technologies described above and it will use the Vue.js framework. Vue is a JavaScript framework for building user interfaces. It builds on top of standard HTML, CSS, and JavaScript and provides a declarative, component-based programming model that helps you efficiently develop user interfaces of any complexity. The two core features of Vue are:

- Declarative Rendering: Vue extends standard HTML with a template syntax that allows us to declaratively describe HTML output based on JavaScript state.
- Reactivity: Vue automatically tracks JavaScript state changes and efficiently updates the DOM when changes happen.

Vue can be used in different ways:

- Enhancing static HTML without a build step
- Embedding as Web Components on any page
- Single-Page Application (SPA)
- Fullstack / Server-Side Rendering (SSR)
- Jamstack / Static Site Generation (SSG)
- Targeting desktop, mobile, WebGL, and even the terminal

[51]

2.2.6. Containers, Docker and Kubernetes

We already talked about the programming languages and the most important programming libraries which we will use for developing our application. In this section we will talk about the most important technologies which we will use in order to deploy and devliver our application.

Containers are executable units of software that package application code along with its libraries and dependencies. They allow code to run in any computing environment, whether it be desktop, traditional IT or cloud infrastructure. Containers take advantage of a form of operating system (OS) virtualization in which features of the OS kernel (for example, Linux namespaces and cgroups, Windows silos and job objects) can be used to isolate processes and control the amount of CPU, memory and disk that those processes can access. More portable and resource-efficient than virtual machines (VMs), containers have become the de facto compute units of modern cloud-native applications. Additionally, containers are critical to the underlying IT infrastructure that powers hybrid multicloud settings—the combination of on-premises, private cloud, public cloud and more than one cloud service from more than one cloud vendor. One way to better understand a container is to examine how it differs from a traditional virtual machine (VM), which is a virtual representation or emulation of a physical computer. A VM is often referred to as a guest, while the physical machine it runs on is called the host. Virtualization technology makes VMs possible. A hypervisor—a small software layer—allocates physical computing resources (for example, processors, memory, storage) to each VM. It keeps each VM separate from others so they don’t interfere with each other. Each VM then contains a guest OS and a virtual copy of the hardware that the OS requires to run, along with an application and its associated libraries and dependencies. VMware was one of the first to develop and commercialize virtualization technology based on hypervisors. Instead of virtualizing the underlying hardware, container technology virtualizes the operating system (typically Linux) so each container contains only the application and its libraries, configuration files and dependencies. The absence of the guest OS is why containers are so lightweight and, thus, faster and more portable than VMs. Containers and virtual machines are not mutually exclusive. For instance, an organization might leverage both technologies by running containers in VMs to increase isolation and security and leverage already installed tools for automation, backup and monitoring [52]. Here are the top advantages of using containers:

- Lightweight: Containers share the machine OS kernel, eliminating the need for a full OS instance per application and making container files small and easy on resources. A container’s smaller size, especially compared to a VM, means it can spin up quickly and better support cloud-native applications that scale horizontally.
- Portable and platform-independent: Containers carry all their dependencies with them, meaning that software can be written once and then run without needing to be re-configured across computing environments (for example, laptops, cloud and on-premises).
- Supportive of modern development and architecture: Due to a combination of their deployment portability and consistency across platforms and their small size, containers are an ideal fit for modern development and application patterns—such as DevOps, serverless and microservices —that are built by using regular code deployments in small increments.
- Improved utilization: Like VMs, containers enable developers and operators to improve CPU and memory utilization of physical machines. Containers go even further because they enable microservices architecture so that application components can be deployed and scaled more granularly. This is an attractive alternative to scaling up an entire monolithic application because a single component is struggling with its load.
- Faster time to market: Containers rely less on system resources, making them faster to manage and deploy than VMs. This feature helps save money and time on application deployment and optimizes time to market.

[52]

The most popular containerization technology nowadays is Docker. Docker enables developers to build, deploy, run, update and manage containers. Docker uses the Linux kernel (the operating system’s base component) and kernel features (like Cgroups and namespaces) to separate processes so they can run independently. Docker essentially takes an application and its dependencies and turns them into a virtual container that can run on any Windows, macOS or Linux-running computer system. Docker is based on a client-server architecture, with Docker Engine serving as the underlying technology. Docker provides an image-based deployment model, making sharing apps simple across computing environments [52]. Docker is an open platform for developing, shipping, and running applications. Docker allows you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production [53]. Docker provides many tools and components, the most important of which is **Docker Engine**. Docker Engine is an open source containerization technology for building and containerizing your applications. Docker Engine acts as a client-server application with: A server with a long-running daemon process dockerd. APIs which specify interfaces that programs can use to talk to and instruct the Docker daemon. A command line interface (CLI) client docker. The CLI uses Docker APIs to control or interact with the Docker daemon through scripting or direct CLI commands. Many other Docker applications use the underlying API and CLI. The daemon creates and manages Docker objects, such as images, containers, networks, and volumes. [54] **Docker Build** is one of Docker Engine's most used features. Whenever you are creating an image you are using Docker Build. Build is a key part of your software development life cycle allowing you to package and bundle your code and ship it anywhere. Docker Build is more than a command for building images, and it's not only about packaging your code. It's a whole ecosystem of tools and features that support not only common workflow tasks but a lso provides support for more complex and advanced scenarios [55]. Finally, when we talk about docker is essential to talk about **Docker Compose** too. Docker Compose is a tool for defining and running multi-container applications. It is the key to unlocking a streamlined and efficient development and deployment experience. Compose simplifies the control of your entire application stack, making it easy to manage services, networks, and volumes in a single, comprehensible YAML configuration file. Then, with a single command, you create and start all the services from your configuration file. Compose works in all environments; production, staging, development, testing, as well as CI workflows. It also has commands for managing the whole lifecycle of your application:

- Start, stop, and rebuild services
- View the status of running services
- Stream the log output of running services
- Run a one-off command on a service

[56]

Since we talked about the fundamental concepts of containers and docker, which enables to build and run containers easily, it's time to talk about a more advanced containerization technology used for more complicated production systems which usually run multiple containers, this is **Kubernetes**. Kubernetes, also known as K8s, is an open source system for automating deployment, scaling, and management of containerized applications. It groups containers that make up an application into logical units for easy management and discovery. Kubernetes builds upon 15 years of experience of running production workloads at Google, combined with best-of-breed ideas and practices from the community [57]. Kubernetes is a portable, extensible, open source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available. Containers are a good way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime. For example, if a container goes down, another container needs to start. Kubernetes provides you with a framework to run distributed systems resiliently. It takes care of scaling and failover for your application, provides deployment patterns, and more. For example:

Kubernetes can easily manage a canary deployment for your system [58]. Some features of kubernetes are:

- Service discovery and load balancing Kubernetes can expose a container using the DNS name or using their own IP address. If traffic to a container is high, Kubernetes is able to load balance and distribute the network traffic so that the deployment is stable.
- Storage orchestration Kubernetes allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more.
- Automated rollouts and rollbacks You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for your deployment, remove existing containers and adopt all their resources to the new container.
- Automatic bin packing You provide Kubernetes with a cluster of nodes that it can use to run containerized tasks. You tell Kubernetes how much CPU and memory (RAM) each container needs. Kubernetes can fit containers onto your nodes to make the best use of your resources.
- Self-healing Kubernetes restarts containers that fail, replaces containers, kills containers that don’t respond to your user-defined health check, and doesn’t advertise them to clients until they are ready to serve.
- Secret and configuration management Kubernetes lets you store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys. You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration.
- Batch execution In addition to services, Kubernetes can manage your batch and CI workloads, replacing containers that fail, if desired.
- Horizontal scaling Scale your application up and down with a simple command, with a UI, or automatically based on CPU usage.
- IPv4/IPv6 dual-stack Allocation of IPv4 and IPv6 addresses to Pods and Services Designed for extensibility Add features to your Kubernetes cluster without changing upstream source code.

[58]

3. Specific Knowledge Assistance Approaches

As we have already described, the goal of this thesis is to create a system that helps its users to find and search existing knowledge. We need to specify with more details our problem domain in order to describe the approaches we will use. There is a single common source of documents for all users. These documents can be of various formats like text, markdown, pdf, html or xml files and we will refer to them as “raw input”. The users should be able to prompt the system by asking questions relative to these documents. The system should be able to provide the users the relative information, by using artificial intelligence technologies.

We will approach the problem with two different methods. For the first method we will create a custom text generation model from scratch. We will train this model with the “raw input” data and so it should be able to generate text according to user prompting. For the first method we will take some existing pre-trained text generation models. These models we will be able to answer multile questions according to their training. Howerver, we will use the RAG technique, which we have described in Section 2.1.4, and we will try to both expand its knowledge with the “raw input” data and at the same to limit its knowledge only to these “raw input” data.

3.1. Custom Text Generation Model Method

In this section we will describe the custom text generation model method in more depth. We will analyze the important steps for our solution and show the most important code snippets of our implementation. Then we list the advantages and dissadvantages of this method.

3.1.1. Method Description

The first step is to read the “raw input” data. For this task we will use the python modules `"langchain_community.document_loaders"` (uses the unstructured library underneath for which we have talked about at Section 2.2.3) in order to read varius documents of different, which exist in a given path `"data_path"`

```
def read_docs(data_path: str) -> list[Document]:
    txt_loader = DirectoryLoader(data_path, glob="**/*.txt", loader_cls=TextLoader,
silent_errors=True, show_progress=True, use_multithreading=True)
    txt_docs = txt_loader.load()
     md_loader = DirectoryLoader(data_path, glob="**/*.md", loader_cls=TextLoader,
silent_errors=True, show_progress=True, use_multithreading=True)
    md_docs = md_loader.load()
                    pdf_loader     =     DirectoryLoader(data_path,     glob="**/
*.pdf", loader_cls=UnstructuredPDFLoader, silent_errors=True, show_progress=True,
use_multithreading=True)
    pdf_docs = pdf_loader.load()
                    html_loader     =     DirectoryLoader(data_path,     glob="**/
*.html", loader_cls=UnstructuredHTMLLoader, silent_errors=True, show_progress=True,
use_multithreading=True)
    html_docs = html_loader.load()
                    xml_loader     =     DirectoryLoader(data_path,     glob="**/
*.xml", loader_cls=UnstructuredXMLLoader, silent_errors=True, show_progress=True,
use_multithreading=True)
    xml_docs = xml_loader.load()
    docs = txt_docs + md_docs + pdf_docs + html_docs + xml_docs
    return docs
```

After we have read all the documents with conantenate them to a single string source. We then create our vocabulary by splitting the string source into unique words using the `word_tokenize` function of module `nltk.tokenize` (from library NLTK for which we talked about at Section 2.2.3). We add a special EOS (End Of Sentence) token to this vocabulary. We sort the vocabulary tokens (words) and we create the encoded vocabulary by simply assigning values for first token to 0, second token to 1, third token to 2, etc.

Then we split the whole string source in stentences using the `sent_tokenize` function of module `nltk.tokenize` . We split each sentence into words using `word_tokenize` and after the end of each sentence we append the EOS token. So we have know transformed the source string into the tokens (words including EOS token). We now create the encoded tokens by using the mapping from the encoded vocabulary we created from the previous step. So we now have a list of

We now create our Xtrain tensor, by using a window of 100 (configurable) encoded tokens. So each row of Xtrain tensor has 100 tokens, and every next row starts from the next token from the start of this row. So first Xtrain row includes token 1 to token 100, second row includes token 2 to token 101, third row includes token 3 to token 103 etc. For each row of Xtrain tensor we add a row at Ytrain tensor with the next token after the last of the row. So first row of Ytrain has token 101, second row of Ytrain has token 102, third row of Ytrain has token 103, etc. With these Xtrain and Ytrain tensors, we will train our model. We select some random sentences and follow the same procedure in order to create Xtest and Ytest tensors for evaluation of our model. It is important to mention that Xtrain and Ytrain are created with all the sentences since we don't want to lose any information of the users' documents, so Xtest and Ytest will not be a very good evaluation since the model will have already see these sentences during its training, maybe in different order. However, this will enable us to be able to determine the performance of our model.

Using PyTorch library (which we talked about at Section 2.2.3) we create a machine learning model. This first layer of the model is an Embedding layer. The Embedding layer is used to store word embeddings and retrieve them using indices. The input to the module is a list of indices, and the output is the corresponding word embeddings [59]. So the embedding layer will help give some meaning to our naive encoding of the tokens. The second layer of the model is an LSTM layer. The LSTM layer is really good at predicting sequential data, so it will enable us to predict the next toekn given the 100 sized window of tokens. Then we add a Dropout layer in order to improve model's performance. The last layer is simply a Linear layer which will have output equal to the number of our unique vocabulary tokens. Here is the basic code of our model definition:

```
class Skalm(nn.Module):
    def __init__(self, skalm_props: SkalmProps, n_vocab: int):
        super(Skalm, self).__init__()
        self.embedding = nn.Embedding(
            num_embeddings=n_vocab,
            embedding_dim=skalm_props.embedding_dim,
            padding_idx=0
        )
        self.lstm = nn.LSTM(input_size=skalm_props.embedding_dim,
                            hidden_size=skalm_props.lstm_hidden_size,
                            num_layers=skalm_props.lstm_num_layers,
                            batch_first=True)
        self.dropout = nn.Dropout(skalm_props.dropout)
```

```python
        self.linear = nn.Linear(skalm_props.lstm_hidden_size, n_vocab)

    def forward(self, x):
        # embedding
        x = self.embedding(x)
        # lstm
        x, _ = self.lstm(x)
        # dropout
        x = self.dropout(x)
        # take only the last output
        x = x[:, -1, :]
        # produce output
        x = self.linear(x)
        return x
```

We now create a training loop in order to train our model using `dataset_train` created by `Xtrain` and `Ytrain` tensors. We are using the Adam optimizer. Adam is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments [60]. We use the CrossEntropyLoss as our loss function. CrossEntropyLoss criterion computes the cross entropy loss between input logits and target and it is useful when training a classification problem with C classes [61]. In our case the number of classes is the number of our unique vocabulary tokens. Using the argmax function we are able to get the token with highest probability to be the next token. In each train epoch with calculate the loss and the accuracy of the model. After each training epoch, we evaluate the model with the `dataset_test` created by `Xtest` and `Ytest` tensors. During the training we save each model with less loss than our current best model for comparison purposes and finally we keep the model with the least loss as our final model.

```python
def train_one_epoch(model: Skalm, loader: DataLoader, optimizer: optim.Adam, loss_fn:
nn.CrossEntropyLoss) -> tuple[float, float]:
    train_epoch_total_loss = 0
    train_epoch_total_correct = 0
    for X_batch, y_batch in loader:
        # Zero your gradients for every batch!
        optimizer.zero_grad()
        # Make predictions for this batch
        y_pred = model(X_batch)
        # Compute the loss and its gradients
        loss = loss_fn(y_pred, y_batch)
        loss.backward()
        # Adjust learning weights
        optimizer.step()
        # calculate
        train_epoch_total_loss += loss.item()
        winners = y_pred.argmax(dim=1)
        train_epoch_total_correct += (y_batch == winners).sum().item() / len(y_batch)

    train_epoch_loss = train_epoch_total_loss / len(loader)
    train_epoch_accuracy = train_epoch_total_correct / len(loader)
    return train_epoch_loss, train_epoch_accuracy
```

```
def validate_one_epoch(model: Skalm, loader_test: DataLoader, loss_fn:
nn.CrossEntropyLoss):
    test_epoch_total_loss = 0
    test_epoch_total_correct = 0
    for X_batch, y_batch in loader_test:
        y_pred = model(X_batch)
        test_loss = loss_fn(y_pred, y_batch)
        test_epoch_total_loss += test_loss.item()
        winners = y_pred.argmax(dim=1)
        test_epoch_total_correct += (y_batch == winners).sum().item() / len(y_batch)

    test_epoch_loss = test_epoch_total_loss / len(loader_test)
    test_epoch_accuracy = test_epoch_total_correct / len(loader_test)
    return test_epoch_loss, test_epoch_accuracy

def train_skallm_lstm(model: Skalm, skalm_config: SkalmConfig, skalm_dir_path: str,
Xtrain: Tensor, Ytrain: Tensor, Xtest: Tensor, Ytest: Tensor) -> tuple[dict[str, Any] |
None, list[float], list[float]]:
    n_epochs = skalm_config.n_epochs
    batch_size = skalm_config.batch_size
    optimizer = optim.Adam(model.parameters(), lr=skalm_config.lr)
    loss_fn = nn.CrossEntropyLoss(reduction="mean")
    dataset_train = TensorDataset(Xtrain, Ytrain)
    dataset_test = TensorDataset(Xtest, Ytest)
    loader_train = DataLoader(dataset_train, shuffle=True, batch_size=batch_size)
    loader_test = DataLoader(dataset_test, shuffle=True, batch_size=batch_size)
    best_model_state_dict = None
    best_loss = np.inf
    train_losses: list[float] = []
    train_accuracy_list: list[float] = []
    test_losses: list[float] = []
    test_accuracy_list: list[float] = []
    for epoch in range(n_epochs):
        model.train(True)
        train_epoch_loss, train_epoch_accuracy = train_one_epoch(model, loader_train,
optimizer, loss_fn)
        train_losses.append(train_epoch_loss)
        train_accuracy_list.append(train_epoch_accuracy)
        # Validation
        model.eval()
        with torch.no_grad():
            test_epoch_loss, test_epoch_accuracy = validate_one_epoch(model, loader_test,
loss_fn)
            test_losses.append(test_epoch_loss)
            test_accuracy_list.append(test_epoch_accuracy)
            if test_epoch_loss <= best_loss:
                best_loss = test_epoch_loss
                best_model_state_dict = model.state_dict()
                save_model(skalm_dir_path, best_model_state_dict, train_losses,
train_accuracy_list, test_losses, test_accuracy_list, epoch)
```

```
    save_model(skalm_dir_path, best_model_state_dict, train_losses, train_accuracy_list, 
test_losses, test_accuracy_list, None)
    return best_model_state_dict, train_losses, test_losses
```

We perform the training only on CPU due to the hardware restrictions of our laptop ("HP Laptop 15s-eq2xxx" with processor "AMD Ryzen 5 5500U") for 5 days using only one book as a user document [62]. We show the evaluation results:

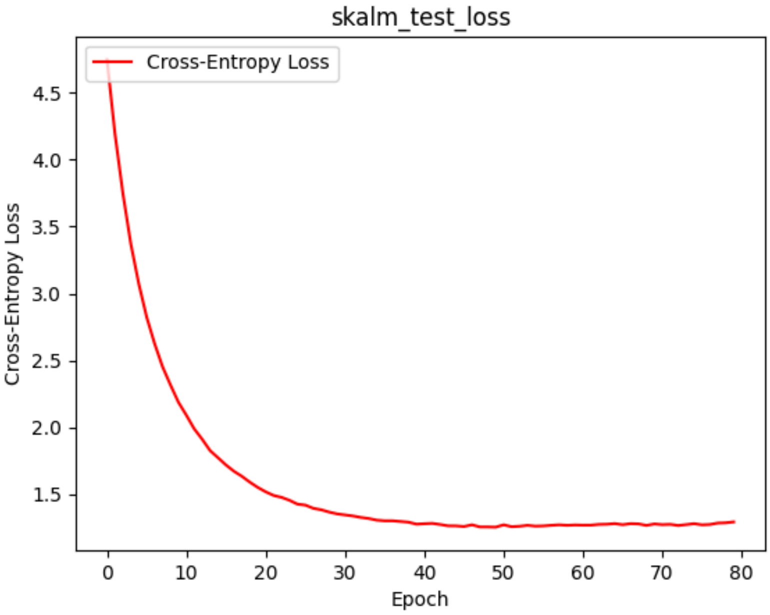

IMAGE 4: SkaLM Test Loss

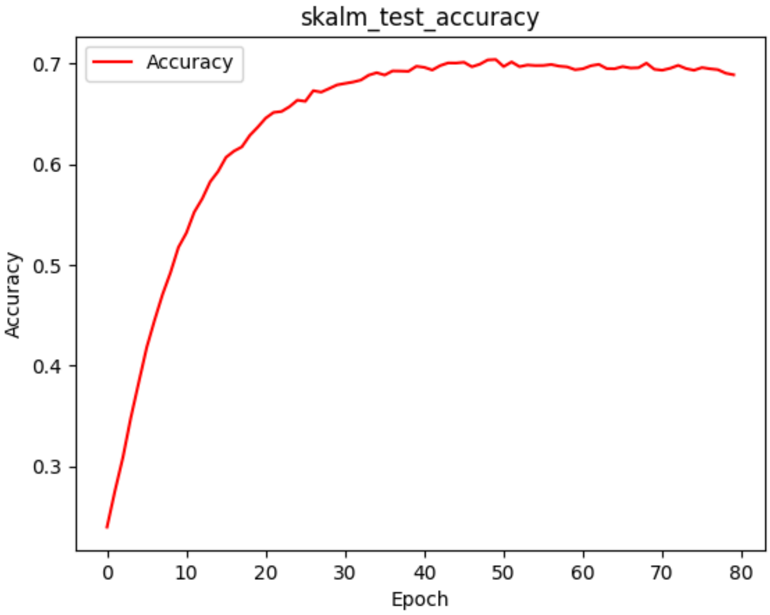

IMAGE 5: SkaLM Test Accuracy

When the user prompts the system with this model then we split the question to sentences and then to encoded tokens maximum of 100 (same as the window we used for training) tokens. If the question is less than 100, then we add a special padding token to the left as many times as needed in order to reach 100 tokens. The system invokes the model in order to predict the next word. The system then shifts the question to the left, replaces the last token with the predicted one and then invokes the model again until it reaches a maxium of generated tokens (defualt 160) or a maximum of EOS tokens (default 6).

3.1.2. Method Advantages and Dissadvantages

We described the process of the method. This method offers very little advantages and is not very suitable for a complete fully functional solution.

Advantages of this method are:

- Independence: Does not rely in any pre-trained text generation models.
- Flexibility: The model creation and training is fully customizable and can be changed to better suit the knowledge field for which the assistant is needed.

Dissadvantages of this method are:

- Needs many sources: The model needs to learn both the language and the knowledge field from the documents, so it needs thousands of documents in order to be able to perform somewhat well.
- Highly consuming: Each time a document is added, removed or edited the model needs to be retrained, which is a very time and resource consuming task.
- Hard to implement: This method needs a dedicated data science team to fine tune and improve the model parameters and layers, in order to be able to achieve a satisfying result for a specific knowledge field.
- Hard to return sources: It's really hard, almost impossible, to return the sources of the relative information with which the answer was constructed.

3.2. Retrieval Augmented Generation (RAG) Method

In this section we will describe the Retrieval Augmented Generation (RAG) method in more depth. We will analyze the important steps for our solution and show the most important code snippets of our implementation. Then we list the advantages and dissadvantages of this method.

3.2.1. Method Description

For the RAG method we will depend on existing pre-trained LLMs. We have described the characteristics of LLMs at Section 2.1.3.

The first step is to read the "raw input" data from the users' documents similar to what we did in the previous method. We will use the same `read_docs` python function we showed before.

We will use class `RecursiveCharacterTextSplitter` from module `langchain.text_splitter` in order to split the "raw input" in to chunks:

```
docs = read_docs(data_path)
# transformation: split the documents into chunks
splitter = RecursiveCharacterTextSplitter(
    chunk_size=128,
    chunk_overlap=8
)
texts = splitter.split_documents(docs)
```

Then we need an embedding model. Embeddings create a vector representation of a piece of text. This is useful because it means we can think about text in the vector space, and do things like semantic search where we look for pieces of text that are most similar in the vector space [63]. We will use the “all-MiniLM-L6-v2” pre-trained model from hugging face, which maps sentences & paragraphs to a 384 dimensional dense vector space and can be used for tasks like clustering or semantic search. [64]. Then we will use the FAISS vector store (which we described in Section 2.2.3) in order to save the embedding documents as a vector store to a local path.

```
embeddings = HuggingFaceEmbeddings(
    model_name=embedding_model_path,
    model_kwargs={'device': 'cpu'},
    encode_kwargs = {'normalize_embeddings': True}
)
db = FAISS.from_documents(texts, embeddings)
db.save_local(vector_store_path, constants.VS_INDEX_NAME)
```

Now when the user is prompting the system we create LangChain chain as of official documentation [65]. For this we create a pipeline of an existing pre-trained model. In our custom python function `create_llm` we support two types of pre-trained models. The first one is using llama.cpp libray we described in Section 2.2.3 for models in gguf format (mainly llama2 and llama3). The second type is using `AutoModelForCausalLM` with `HuggingFacePipeline` which supports most text generation models which have been uploaded in hugging face hub together with their configuration. We load the vector store which we stored in the previous steps. We create a more complicated chain in order to be able to return the relevant source together with the answer as per official documentation [66]. The result consists of the `question` , the `answer` and the `context` which is a list with the relevant Documents of the chunks in which the model found the answer. More advanced models like llama2 and llama3 accept system prompting which allows the users to instruct the LLM in which way to construct the final answer.

```
def create_llm(llm_model_path: str, model_type: str, temperature: int, top_p: int):
    context_length = 512
    max_tokens = 160
    batch_size = 512
    last_n_tokens = 8
    repetition_penalty = 1.1
    if model_type == 'llamacpp':
        # Callbacks support token-wise streaming
        callback_manager = CallbackManager([StdOutCallbackHandler()])
        llm = LlamaCpp(
            model_path=llm_model_path,
            temperature=temperature,
            max_tokens=max_tokens,
            last_n_tokens_size=last_n_tokens,
            top_p=top_p,
            callback_manager=callback_manager,
            verbose=True,  # Verbose is required to pass to the callback manager
            client=None,
            n_ctx=context_length,
            n_parts=-1,
            seed=-1,
            f16_kv=True,
            logits_all=False,
            vocab_only=False,
            use_mlock=False,
            n_threads=None,
            n_batch=batch_size,
            n_gpu_layers=None,
            suffix=None,
            logprobs=None,
            repeat_penalty=repetition_penalty
        )
        return llm
    elif model_type == 'huggingface':
        tokenizer = AutoTokenizer.from_pretrained(llm_model_path)
        model = AutoModelForCausalLM.from_pretrained(llm_model_path, device_map='cpu')
                pipe  =  pipeline("text-generation",  model=model,  tokenizer=tokenizer,
max_length=context_length)
        llm = HuggingFacePipeline(pipeline=pipe)
        return llm
    else:
        raise Exception(f"unsuported model_type {model_type}")


def create_chain(vector_store_path: str, embedding_model_path: str, llm_model_path: str,
prompt_template: str, model_type: str, temperature: int, top_p: int):

    # load the language model
    llm = create_llm(llm_model_path, model_type, temperature, top_p)
    # load the interpreted information from the local database
    embeddings = get_embeddings(embedding_model_path)
       db  =  FAISS.load_local(vector_store_path,  embeddings,  constants.VS_INDEX_NAME,
allow_dangerous_deserialization=True)
    # prepare a version of the llm pre-loaded with the local content
    retriever = db.as_retriever(search_kwargs={'k': 9})
    prompt = PromptTemplate(
        template=prompt_template,
        input_variables=['context', 'question']
    )

    def format_docs(docs):
        return "\n\n".join(doc.page_content for doc in docs)

    rag_chain_from_docs = (
        RunnablePassthrough.assign(context=(lambda x: format_docs(x["context"])))
        | prompt
        | llm
        | StrOutputParser()
    )
    rag_chain_with_source = RunnableParallel(
        {"context": retriever, "question": RunnablePassthrough()}
    ).assign(answer=rag_chain_from_docs)
    return rag_chain_with_source


qa_chain = create_chain(vector_store_path, embedding_model_path, llm_model_path,
prompt_template, model_type, temperature, top_p)
output = qa_chain.invoke(question)
invoke_output = InvokeOutput.from_output_dict(output)
```

3.2.2. Method Advantages and Dissadvantages

We described the process of the RAG method. This method offers many advantages with only a few dissadvantages and therefore it is recomended for a full functional system.

Advantages of this method are:

- Needs only relative sources: The model already understand the needed language, so it needs access only to the relevant documents for our desired knowledge field.
- Adaptable: New, deleted or altered documents demand only the vector store to be recreated, so the systemd can adapt to changes relatively quickly.
- East to implement: This method can be implemented easily with the dependency on the libraries we use without any deep advanced knowledge on how LLMs work.
- Able to return sources: It’s really easy to return the relevant sources in which the answer was found.
- Prompting: It’s easy to specify system prompting with advanced LLMs which support it, in order to change the style or even the allowed or dissallowed content of the answers.

Dissadvantages of this method are:

- Dependency on external LLMs: It requires external pre-trained LLMs. While nowadays there are plenty of those and new ones are created almost every year, it is not always clear how they are trained and what biases they have.
- Inflexibility: The model structure and training procedure cannot be altered (sometimes it can be extended with additional sources though).

4. System Architecture

Now that we have explained the two methods on which our solutions is based upon, we are ready to describe the system architecture of our complete solution.

4.1. System Components

We have develop 4 basic components for our application. The `ska_llm` component, which contains a library and a command line application, is implemented in python and it is responsible for all document processing and machine learing tasks. The `ska_cli` command line application is implemented in rust and is responsible for various administration tasks and uses the `ska_llm` internally to expose a more friendly command line interface for the machine learning tasks needed to be performed by the admin. The `ska_server` component, implemented in rust as well, exposes a REST API, which enables users to perform the various tasks of the application, like prompting the LLMs. Finally, the `ska_frontend` materializes the Graphical User Interface with which the users can interact with.

Apart from the four components we developed, we utilize three extra components for our complete solution. The first one is Keycloak [67], an open source Identity and Access Management solution which enables our users to login and to be authenticated and authorized to perform specific actions of our application. The second one is PostgreSQL [68], an advanced open source relational database which our application uses for storing and fetching data. The final component is an `api gateway` which will lay ahead of our services and redirect the web traffic to the needed service for the task.

In the following diagram, we show the fundamental Specific Knowledge Assistant (SKA) System Components. With green color we have marked the components that we develop. With burgundy color we have marked the extra components that we utilize and configure them in order to complete our system.

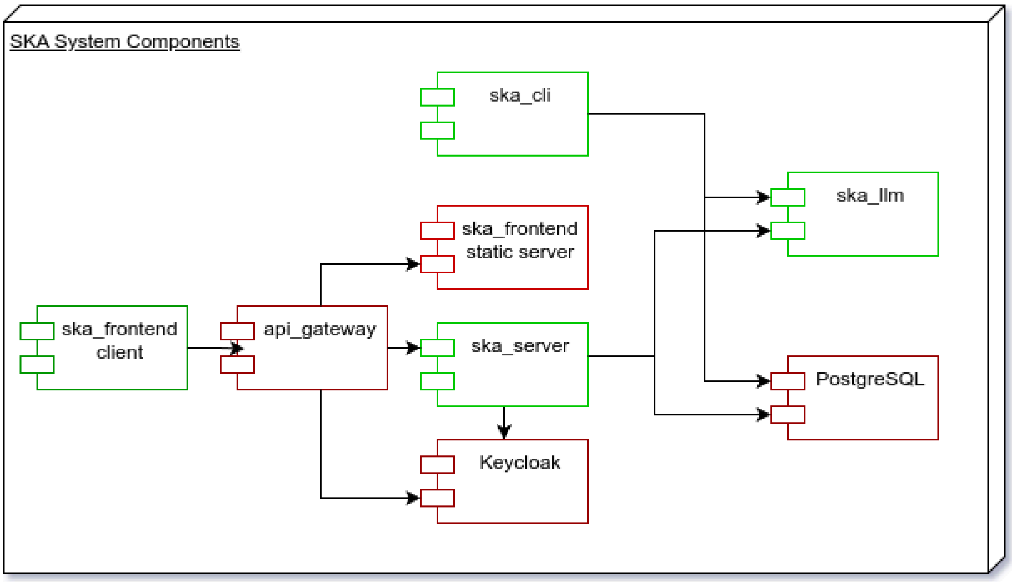

IMAGE 6: SKA System Components

As we already said, the `ska_llm` component is implemented in python and it is responsible for all document processing and machine learing tasks. It supports 5 crucial for our application operations which are using the libraries we described in Section 2.2.3. The operation `download_llm` downloads some given models from hugging face hub [35]. The operation `rag_prepare` performs the prepartation for the RAG method that we discussed in Section 3.2.1, including reading users' documents and storing their chunks in vector store. The operation `rag_invoke` invokes the given LLM and answers the given question based on the context of the relevant chunks found in the vector store. The operation `create_llm` creates and trains the SKA text generation model we described in Section 3.1.1. The operation `invoke_skalm` invokes our custom generation model and answers the question with the procedure we described in Section 3.1.1. We show a high level diagram of these basic operations:

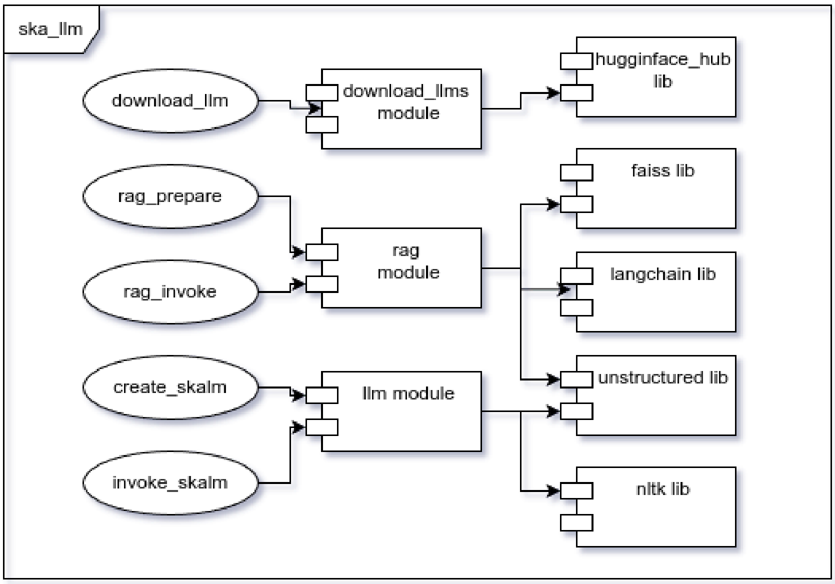

IMAGE 7: SKA LLM operations

We have already said that the `ska_cli` command line application is implemented in rust, it is responsible for various administration tasks and it uses the `ska_llm` internally. This cli component expose a friendly command line interface so that the admin can perform various operations. For the command line interface and argument parsing we will depend on `clap` library. For all datbase access we depend on the SeaORM library which we have describe in Section 2.2.4. There are two top level commands, the `command_db` and `command_model` . The first command, `command_db` , is responsible for administration tasks that affect our PostgreSQL database. It has a single child command `command_migrate` which migrates the database schema if there is a new schema version available. The second top level command, `command_model` , is responsible for all the machine learning tasks. We don't use any relevant rust libraries for these tasks, but the core functionality is handled by our `ska_llm` component as we have explained before. It has 5 child commands. The first command `command_download` downloads the needed machine learning models form hugging face hug [35]. The pre defined list with supported models currently are:

- all-MiniLM-L6-v2 [64]: A sentence-transformers model which maps sentences & paragraphs to a 384 dimensional dense vector space. It is used as our embedding model when we are storing/ searching context chunks in/from our vectore store.
- Llama-2-7B-Chat-GGUF [69]: A Llama-2 model quantized for better invoking times performance, but with less accuracy.
- Meta-Llama-3-8B-Instruct-GGUF [70]: A Llama-3 model quantized for better invoking times performance, but with less accuracy.

The second command `command_insert` inserts the models from the temporary downloaded location to a persisted location that can be read by our application. The commands `command_create_skalm` and `command_rag_prepare` are essentially wrappers around the equivalent operations of our `ska_llm` module. They also perform some extra checks and validations to disallow unwanted consequences. The command `command_rag_invoke` performcs validations and uses internally either `rag_invoke` or `invoke_skalm` operations of `ska_llm` components dependening on the selected model. The following is a high level diagram showing the basic operations of `ska_cli` and the most important modules they are using.

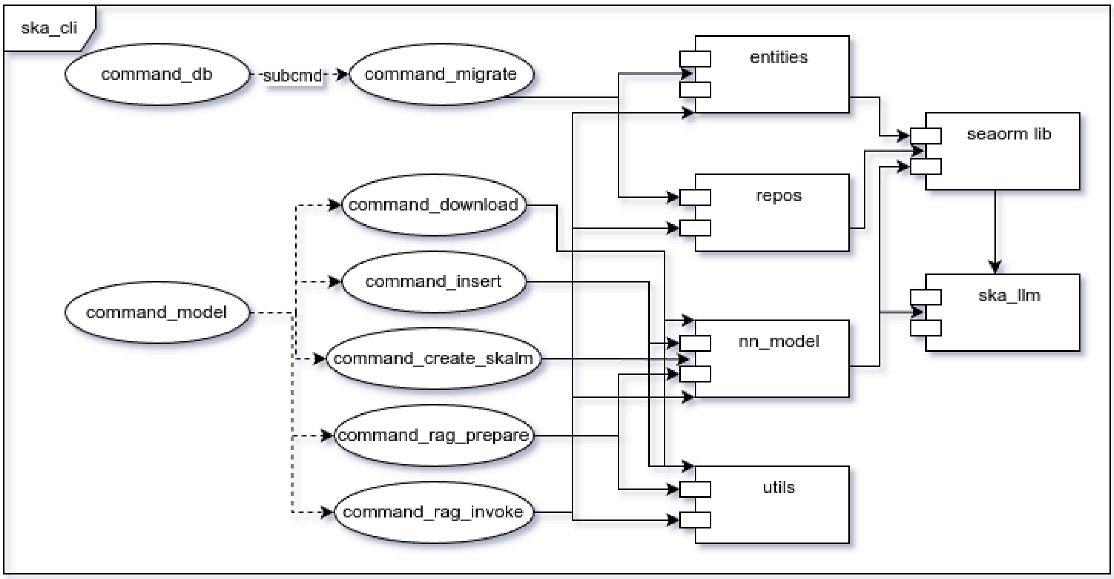

IMAGE 8: SKA CLI operations

The `ska_server` component is implemented in rust, it exposes a REST API with some http endpoints which will be called by the `ska_frontend` component (we will describe in a later paragraph). It uses the same code base as the `ska_cli` component in order to have consistent business logic and be able to share domain relevant modules like `entities` and `repos` . For the http api construction we depend on the `axum` library which we discussed in Section 2.2.4. All the http endpoints are protected and need an authenticated user with required permissions for allowed access. The authentication is performed by implementing an `auth_middleware` which is used by all endpoints. Each endpoint demands different user roles for a user to have in order to be allowed to access it. The there are two top level routes `route_auth` and `route_assistant` . The first `route_auth` has only on operation `app_login` and is called when a user is logged in the app in order to update the user details from the user pool to our database. Then second top level endpoint route `route_assistant` enables three basic categories of operations. The first category `fetch` is about fetching user data like chats and historical messages. The second category `chat CRUD` is about CRUD (Create Read Upate Delete) operations on user's chats, which allows the user to manage his chats. The last `ask question` is about the user asking a question to a selected chat which uses a defined LLM. This operation perfromes various validations and uses our `ska_llm` component internally as well in order to invoke the required LLM. We show a simple high level diagram of `ska_server` component.

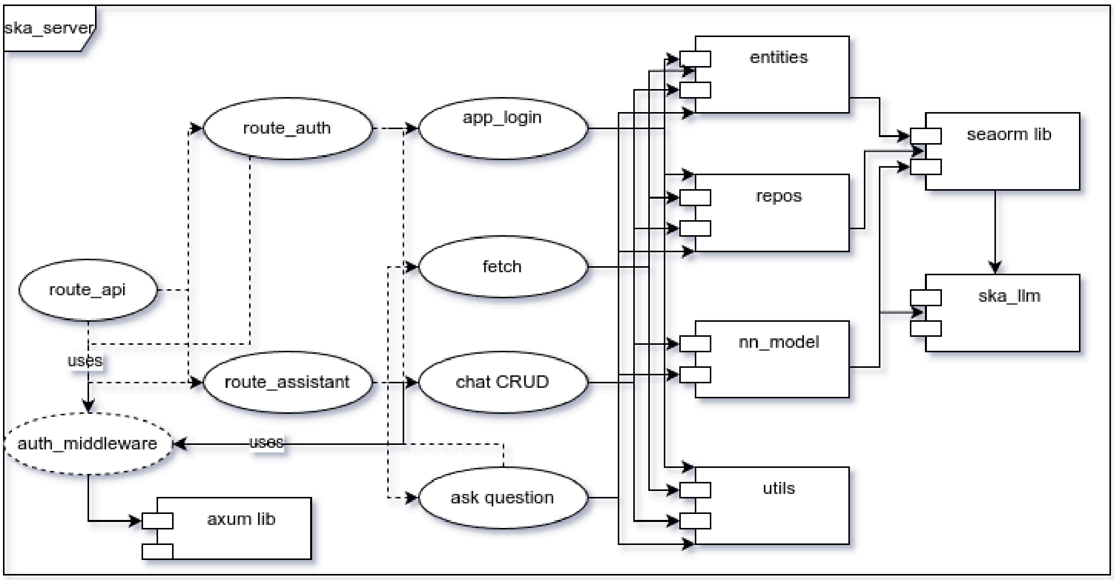

IMAGE 9: SKA Server operations

The `ska_frontend` component materializes the Graphical User Interface with which the users can interact with. It actually consistfs of two sub components. The first one is `ska_frontend client` which contains all the built and bundled final needed files like html, css, javascript, images, etc. The second sub component `ska_frontend static server` is a simple http server, implemented by using a simply configured nginx instance, which serves all the files needed for the `ska_frontend client` to run in the users' browsers. The `ska_frontend client` is what we call a Single Page Application (SPA) [46] so the pages can contain complex logic in the client side and dynamically change their content without fetching new complete html pages from a server for any single request. The implementation is based on the Vue.js library [51]. The multiple pages can be achieved in the client side by using the `Vue router` library [71]. The client is using the "Asynchronous JavaScript and XML" (AJAX) [72] technique in order to fetch data from our `ska_server` by utilizing the `axios` library [73]. For the authentication and authorization to work, we are using the `oidc-client-ts` library [74] which communicates with the `Keycloak` instance in order to fetch the needed tokens (we will talk about it with more details when we will describe Keycloak). The client provides three top level pages from which users can perform their operations. The `page_home` is the first page that is shown to a user. It shows a short description about the app and its content changes slightly depending on whether the user is logged in or not. The `page_account` enables a logged in user to see his account details as well as to edit them. The `page_assistant` is the most important page as it enables a logged in user to use the Specific Knowledge Assistant. There a user can create, edit or delete a chat with a selected LLM. The user can then ask questions in this chat and fetch historical questions and answers of existing chats. The following diagram shows a high level overviews of `ska_frontend` :

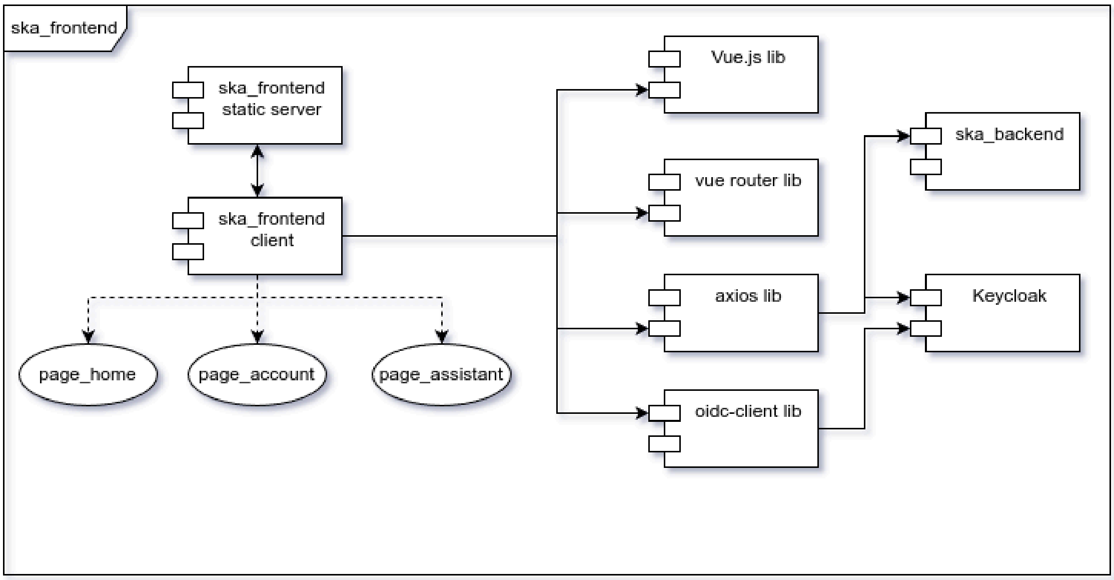

IMAGE 10: SKA Frontend operations

The `PostgreSQL` database is used in order to persist users' data. Specifically we use the table `users` to store each user who logs in to our application. We update their details in each login. Each user can have multiple chats which we store to table `user_chat` . Each chat can have multiple messages some of them are "ANSWER" type and some of them are "QUESTION" type. We store the messages to table `chat_message` . Here is the Entity Relationship Diagram (ERD) of our database:

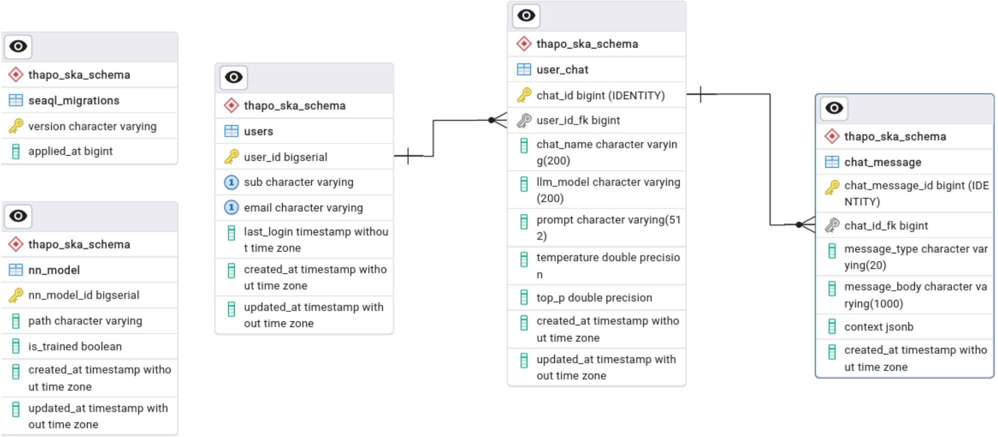
thapo_ska_schema
seaql_migrations
version character varying
applied_at bigint
thapo_ska_schema
nn_model
nn_model_id bigserial
path character varying
is_trained boolean
created_at timestamp without time zone
updated_at timestamp without time zone
thapo_ska_schema
users
user_id bigserial
sub character varying
email character varying
last_login timestamp without time zone
created_at timestamp without time zone
updated_at timestamp without time zone
thapo_ska_schema
user_chat
chat_id bigint (IDENTITY)
user_id_fk bigint
chat_name character varying(200)
llm_model character varying (200)
prompt character varying(512)
temperature double precision
top_p double precision
created_at timestamp without time zone
updated_at timestamp without time zone
thapo_ska_schema
chat_message
chat_message_id bigint (IDENTITY)
chat_id_fk bigint
message_type character varying(20)
message_body character varying(1000)
context jsonb
created_at timestamp without time zone

IMAGE 11: SKA Database ERD

The keycloak component is used as a user pool and as a user authentication and authorization utility. For this we have created a specifc SKA Realm which contains the available users together with their assigned roles. The supported roles are:

- SKA_ADMIN: role that practically allows access to all endpoints
- SKA_USER: role that allows a user to edit his account info and use the assistant functionality.
- SKA_GUEST: not used in current version but reserved in case of wanted future functionality additions.

In order to authorize and authenticate users in `ska_frontend` and protect our endpoints in `ska_server` components we use the OpenID Connect “Authorization Code flow”, which follows essentially the Oauth2 “Authorization Code flow with PKCE”. The steps that are performed for this authorization flow are the following

- The user clicks logins in `ska_frontend` .
- `ska_frontend` client using `oidc-client-ts` library creates a cryptographically-random code_verifier and from this generates a code_challenge and redirects user to Keycloak login page along with the code_challenge.
- Keycloak creates a cryptographically-random code_verifier and from this generates a code_challenge.
- The user authenticates using his credentials.
- Keycloak stores the code_challenge and redirects the user back to the application with an authorization code, which is good for one use.
- `ska_frontend` client using `oidc-client-ts` sends this code and the code_verifier to a keycloak endpoint
- Keycloak verifies the code_challenge and code_verifier and responds with an ID token and access token, and a refresh token.
- `ska_frontend` client passes the access token to the Authorization header in every request to `ska_server` using `axios` library.

The following example from frameworks.readthedocs.io can clarify the procedure, in which we can replace the “Angular Client” with our `ska_frontend client` component.

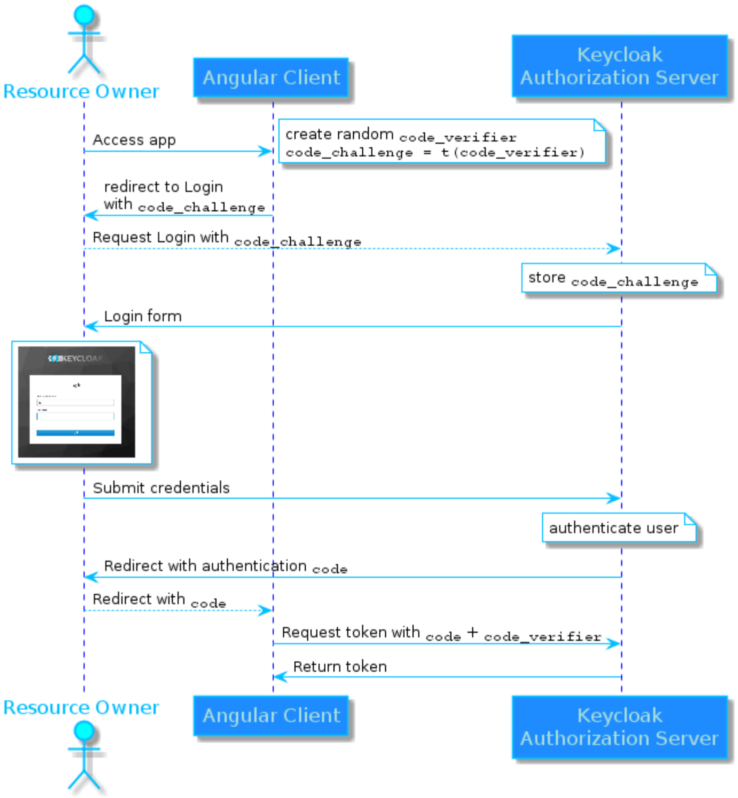
Resource Owner
Angular Client
Keycloak Authorization Server
Access app
create random code_verifier
code_challenge = t(code_verifier)
redirect to Login with code_challenge
Request Login with code_challenge
store code_challenge
Login form
Submit credentials
authenticate user
Redirect with authentication code
Redirect with code
Request token with code + code_verifier
Return token
Resource Owner
Angular Client
Keycloak Authorization Server

IMAGE 12: Keycloak Oauth2 Authorization Code flow with PKCE [75]

The `api gateway` is implemented by either using a nginx [76] reverse proxy or a kubernetes Ingress [77] depending on the deployment method. All the incoming http requests reache this componet at first and then they get redirected to the desired service. This enables us to have a single component which can be configured with rules. Additionally, we have the benefit to serve the whole application under a single owned domain name and redirect the traffic based on http paths. Specifically we configure the following http paths redirections:

- "/app" to `ska_frontend` : any http path that starts with "/app" is redirected to `ska_frontend` service
- "/backend" to `ska_server` : any http path that starts with "/backend" is redirected to `ska_server` service
- "/iam" to `Keycloak` :: any http path that starts with "/iam" is redirected to `Keycloak` service

4.2. Development Lifecycle and Deployment

We talked about our fundamental System components and modules. Now we will briefly talk about the development lifecycle of the systemd and its deployment methods.

The code of our application exists in a git repostiory at https://github.com/ThanosApostolou/thapo-ska. There are 3 fundamental branches. New code are directly pushed into `main` branch or merged by some other feature branch into `main` branch. When chagnes of multiple commits have been tested enough locally, then the `main` branch is merged into `dev` branch. After multiple merges in `dev` branch, when we are ready for a release we merge `dev` branch into `prod` branch.

In alignment with the git branches, the application can run with 3 differnet environments:

- `local` environment
- `dev` environment

- `prod` environment

Each environment specifiies different configuration for our services (e.g. about which url the frontend uses, which url the backend uses, which database schema to use, etc…).

The application is continuously deployed by use CI/CD (Continuous Integration and Continuous Delivery) practices [78]. To achieve this we use Jenkins [79], an open source automation server. In more details:

- `main` : `main` branch is tested only locally, so we don't deploy anything for this.
- `dev` : Whenever changes are merged to `dev` branch, a jenkins pipeline is executed. The basic pipeline tasks are:
  - build all our services as docker images
  - push the built docker images to our private docker repository
  - deploy and start the application with `dev` environment at our private server, using docker compose (see Section 2.2.6).

  The `dev` application is targeting a private port which can be accessed only by our local network or by using vpn. This environment's purpose is to have a fully deployed system for testing purposes which is not affected by local changes.
- `prod` : Whenever changes are merged to `prod` branch, a jenkins pipeline is executed. The basic pipeline tasks are:
  - build all our services as docker images
  - push the built docker images to our private docker repository
  - deploy and start the application with `prod` environment at a k3s [80] kubernetes instance (see Section 2.2.6) installed in our private server. For easier management of all the kubernetes resources we need to create we use Helm [81].

  The `prod` application is targeting a port which can be accessed publicly fron the internet by all the users. Currently the domain our application is using is the https://thapo-ska.thapo.org/app, but it is possibly to change after the completion of this thesis.

5. Usage and Execution of the Application

We talked about the System Architecture of our application, now we will show real execution of our application and its results.

5.1. CLI Usage

5.2. GUI Usage

6. Conclusions and Future Work

TODO

Bibliography

[1] H. Face, “Text Generation.” [Online]. Available: https://huggingfagce.co/tasks/text-generation

[2] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th Global Edition. Pearson, 2021.

[3] Wikipedia, “Artificial intelligence.” [Online]. Available: https://en.wikipedia.org/wiki/Artificial_intelligence

[4] Wikipedia, “Machine Learning.” [Online]. Available: https://en.wikipedia.org/wiki/Machine_learning

[5] GeeksforGeeks, “Types of Machine Learning.” [Online]. Available: https://www.geeksforgeeks.org/types-of-machine-learning

[6] lakeFS, “Machine Learning Components: Elements & Classifications.” [Online]. Available: https://lakefs.io/blog/machine-learning-components/

[7] J. Holdsworth and M. Scapicchio, “Deep learning.” [Online]. Available: https://www.ibm.com/topics/deep-learning

[8] Amazon, “Deep learning.” [Online]. Available: https://aws.amazon.com/what-is/deep-learning/

[9] R. Khalkar, A. S. Dikhit, and A. Goel, “Handwritten Text Recognition using Deep Learning (CNN & RNN),” *International Advanced Research Journal in Science, Engineering and Technology*, 2021, [Online]. Available: https://www.researchgate.net/publication/353939315_Handwritten_Text_Recognition_using_Deep_Learning_CNN_RNN

[10] javatpoint, “Applications of Machine learning.” [Online]. Available: https://www.javatpoint.com/applications-of-machine-learning

[11] GeeksforGeeks, “Applications of Machine learning.” [Online]. Available: https://www.geeksforgeeks.org/machine-learning-introduction/

[12] C. Staff, “10 Machine Learning Applications.” [Online]. Available: https://www.coursera.org/articles/machine-learning-applications

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Artificial Intelligence: Deep learning - adaptive computation and machine learning*. 2016.

[14] K. Martineau, “What is generative AI.” [Online]. Available: https://research.ibm.com/blog/what-is-generative-AI

[15] Cloudflare, “What is a large language model (LLM).” [Online]. Available: https://www.cloudflare.com/learning/ai/what-is-large-language-model/

[16] Ruman, “Setting Top-K, Top-P and Temperature in LLMs.” [Online]. Available: https://rumn.medium.com/setting-top-k-top-p-and-temperature-in-llms-3da3a8f74832

[17] A. Verma, “Understanding temperature, top_p, top_k, logit_bias in LLM parameters.” [Online]. Available: https://aviralrma.medium.com/understanding-llm-parameters-c2db4b07f0ee

[18] K. Talamadupula, “A Guide to Quantization in LLMs.” [Online]. Available: https://symbl.ai/developers/blog/a-guide-to-quantization-in-llms/

[19] LangChaing, “Build a Retrieval Augmented Generation (RAG) App.” [Online]. Available: https://python.langchain.com/v0.2/docs/tutorials/rag/

[20] Amazon, “What is Retrieval-Augmented Generation.” [Online]. Available: https://aws.amazon.com/what-is/retrieval-augmented-generation/

[21] Wikipedia, “Python (programming language).” [Online]. Available: https://en.wikipedia.org/wiki/Python_(programming_language)

[22] Amazon, “What is Python.” [Online]. Available: https://aws.amazon.com/what-is/python/

[23] GeeksforGeeks, “Python Language advantages and applications.” [Online]. Available: https://www.geeksforgeeks.org/python-language-advantages-applications/

[24] Wikipedia, “Rust (programming language).” [Online]. Available: https://en.wikipedia.org/wiki/Rust_(programming_language)

[25] Rust, “Rust.” [Online]. Available: https://www.rust-lang.org/

[26] C. Mittal, “10 Best Use Cases of Rust Programming Language in 2023.” [Online]. Available: https://medium.com/@chetanmittaldev/10-best-use-cases-of-rust-programming-language-in-2023-def4e2081e44

[27] M. Tawfik, “Rust Language: Pros, Cons, and Learning Guide.” [Online]. Available: https://medium.com/@apicraft/rust-language-pros-cons-and-learning-guide-594e8c9e2b7c

[28] N. Project, “Natural Language Toolkit.” [Online]. Available: https://www.nltk.org/

[29] N. Project, “Installing NLTK Data.” [Online]. Available: https://www.nltk.org/data.html

[30] Wikipedia, “PyTorch.” [Online]. Available: https://en.wikipedia.org/wiki/PyTorch

[31] N. Corporation, “PyTorch.” [Online]. Available: https://www.nvidia.com/en-eu/glossary/pytorch/

[32] LangChain, “Introduction.” [Online]. Available: https://python.langchain.com/v0.2/docs/introduction/

[33] LangChain, “LLMs.” [Online]. Available: https://python.langchain.com/v0.1/docs/modules/model_io/llms/

[34] LangChain, “Vector stores.” [Online]. Available: https://python.langchain.com/v0.1/docs/modules/data_connection/vectorstores/

[35] HuggingFace, “Hugging Face Hub documentation.” [Online]. Available: https://huggingface.co/docs/hub/index

[36] G. Gerganov, “llama.cpp.” [Online]. Available: https://github.com/ggerganov/llama.cpp

[37] G. Gerganov, “ggml.” [Online]. Available: https://github.com/ggerganov/ggml/blob/master/README.md

[38] ivanov, M. Brett, and mdboom2, “matplotlib.” [Online]. Available: https://pypi.org/project/matplotlib/

[39] U. Technologies, “unstructured.” [Online]. Available: https://pypi.org/project/unstructured/

[40] M. Douze and L. Hosseini, “faiss.” [Online]. Available: https://pypi.org/project/unstructured/

[41] E. Page and K. K., “clap.” [Online]. Available: https://github.com/clap-rs/clap/blob/master/README.md

[42] C. Lerche, “tokio.” [Online]. Available: https://github.com/tokio-rs/tokio/blob/master/README.md

[43] D. Pedersen and J. Platte, “axum.” [Online]. Available: https://github.com/tokio-rs/axum/blob/main/README.md

[44] B. Chan and C. Tsang, “SeaORM.” [Online]. Available: https://github.com/SeaQL/sea-orm/blob/master/README.md

[45] SeaQL.org, “Database Connection.” [Online]. Available: https://www.sea-ql.org/SeaORM/docs/install-and-config/connection/

[46] M. contributors, “SPA (Single-page application).” [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/SPA/

[47] M. contributors, “CSS: Cascading Style Sheets.” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/CSS

[48] M. contributors, “JavaScript.” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[49] M. contributors, “TypeScript.” [Online]. Available: hthttps://developer.mozilla.org/en-US/docs/Glossary/TypeScript

[50] M. contributors, “HTML: HyperText Markup Language.” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML

[51] E. You, “Introduction.” [Online]. Available: https://vuejs.org/guide/introduction

[52] S. Susnjara and I. Smalley, “What are containers?.” [Online]. Available: https://www.ibm.com/topics/containers

[53] D. Inc, “Get Docker.” [Online]. Available: https://docs.docker.com/get-docker/

[54] D. Inc, “Docker Engine overview.” [Online]. Available: https://docs.docker.com/engine/

[55] D. Inc, “Overview of Docker Build.” [Online]. Available: https://docs.docker.com/build/

[56] D. Inc, “Docker Compose overview.” [Online]. Available: https://docs.docker.com/compose/

[57] T. K. Authors, “kubernetes.” [Online]. Available: https://kubernetes.io/

[58] T. K. Authors, “Overview.” [Online]. Available: https://kubernetes.io/docs/concepts/overview

[59] P. Contributors, “Embedding.” [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html

[60] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization.” [Online]. Available: https://arxiv.org/abs/1412.6980

[61] P. Contributors, “CrossEntropyLoss.” [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

[62] E. Evans, *Domain Driven Design Tackling Complexity in the Heart of Software*, 1st ed. Addison-Wesley, 2004.

[63] LangChain, “Text embedding models.” [Online]. Available: https://python.langchain.com/v0.1/docs/modules/data_connection/text_embedding/

[64] N. Reimers, O. Espejel, P. Cuenca, and T. Aarsen, “all-MiniLM-L6-v2.” [Online]. Available: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

[65] LangChain, “Chains.” [Online]. Available: https://python.langchain.com/v0.1/docs/modules/chains/

[66] LangChain, “Returning sources.” [Online]. Available: https://python.langchain.com/v0.1/docs/use_cases/question_answering/sources/

[67] K. Authors, “Keycloak.” [Online]. Available: https://www.keycloak.org/

[68] T. P. G. D. Group, “PostgreSQL.” [Online]. Available: https://www.postgresql.org/

[69] T. Jobbins, “Llama-2-7B-Chat-GGUF.” [Online]. Available: https://huggingface.co/TheBloke/Llama-2-7B-Chat-GGUF

[70] P. Bendus, A. Kotliar, and C. Crowley, “Meta-Llama-3-8B-Instruct-GGUF.” [Online]. Available: https://huggingface.co/SanctumAI/Meta-Llama-3-8B-Instruct-GGUF

[71] E. You and E. S. M. Morote, “Vue Router.” [Online]. Available: https://router.vuejs.org/

[72] M. contributors, “Ajax.” [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/AJAX

[73] J. J. "Jake" Sarjeant and M. Zabriskie, “Axios.” [Online]. Available: https://axios-http.com/

[74] P. Luginbühl, “oidc-client-ts.” [Online]. Available: https://authts.github.io/oidc-client-ts/

[75] K. Authors, “Keycloak OAuth2 PKCE.” [Online]. Available: https://frameworks.readthedocs.io/en/latest/spring-boot/spring-boot2/keycloakOAuth2PKCE.html

[76] nginx, “nginx.” [Online]. Available: https://nginx.org/en/

[77] T. K. Authors, “INgress.” [Online]. Available: https://kubernetes.io/docs/concepts/services-networking/ingress/

[78] Wikipedia, “CI/CD.” [Online]. Available: https://en.wikipedia.org/wiki/CI/CD

[79] Jenkins, “Jenkins.” [Online]. Available: https://www.jenkins.io/

[80] K. P. Authors, “K3s.” [Online]. Available: https://k3s.io/

[81] H. Authors, “Helm.” [Online]. Available: https://helm.sh/