



**CZECH TECHNICAL
UNIVERSITY
IN PRAGUE**

F3

**Faculty of Electrical Engineering
Department of Computer Science**

Master's Thesis

Natural language processing in medicine using retrieval augmented generation

Bc. Štěpán Bořek
Open Informatics

May 2024

Supervisor: doc. Ing. Daniel Novák, Ph.D.



MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Bořek Štěpán** Personal ID number: **483841**
Faculty / Institute: **Faculty of Electrical Engineering**
Department / Institute: **Department of Computer Science**
Study program: **Open Informatics**
Specialisation: **Software Engineering**

II. Master's thesis details

Master's thesis title in English:

Natural language processing in medicine using retrieval augmented generation

Master's thesis title in Czech:

Zpracování přirozeného jazyka v medicíně pomocí retrieval augmented generation

Guidelines:

The objective is to create a support system specifically designed for social service workers that will improve the effectiveness of their digital interactions with clients.

1. Acquaint yourself with medical natural language processing techniques, particularly employing advanced large language models like Llama 2 or Mistral.
2. Utilize the Retrieval Augmented Generation (RAG) approach to enhance the precision of language model outputs.
3. Create and execute an API for managing a vector database named Chroma, which enables semantic searches and the automatic processing of document embeddings in various file formats such as PDF, DOCX, and TXT.
4. Conduct an assessment of the RAG framework by analyzing its performance across a sample of 10 conversational dialogues. Then, compare these results to the outcomes of traditional methods that do not incorporate the RAG approach.

Bibliography / sources:

Kyo-Joong Oh, DongKun Lee, ByungSoo Ko, Ho-Jin Choi, A Chatbot for Psychiatric Counseling in Mental Healthcare Service Based on Emotional Dialogue Analysis and Sentence Generation, 18th IEEE International Conference on Mobile Data Management (MDM), 2017
Michele Tanana, A Comparison of Natural Language Processing Methods for Automated Coding of Motivational Interviewing, Journal of substance abuse treatment, 2015
Bang Yang and Asif Raza and Yuexian Zou and Tong Zhang, Customizing General-Purpose Foundation Models for Medical Report Generation
Jiang, L.Y., Liu, X.C., Nejatian, N.P. et al. Health system-scale language models are all-purpose prediction engines. Nature 619, 357–362 (2023).
Gao, Yunfan, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang and Haofen Wang. Retrieval-Augmented Generation for Large Language Models: A Survey
Patrick Lewis and Ethan Perez and Aleksandra Piktus and Fabio Petroni and Vladimir Karpukhin and Naman Goyal and Heinrich Küttler and Mike Lewis and Wen-tau Yih and Tim Rocktäschel and Sebastian Riedel and Douwe Kiela, Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks
Chroma - the AI-native open-source embedding database, <https://www.trychroma.com/>

Name and workplace of master's thesis supervisor:

doc. Ing. Daniel Novák, Ph.D. Analysis and Interpretation of Biomedical Data FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **14.02.2024** Deadline for master's thesis submission: _____

Assignment valid until: **21.09.2025**

doc. Ing. Daniel Novák, Ph.D.
Supervisor's signature

Head of department's signature

prof. Mgr. Petr Páta, Ph.D.
Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt

Student's signature

Acknowledgement / Declaration

I would like to thank my supervisor, doc. Ing. Daniel Novák, Ph.D., for his guidance, consultation and fair approach to work. I would also like to thank the rest of our research team for their inputs, the following: Klára Losenická, Cheng Kang, MSc., Bc. Fabián Bodnár, Ing. Patrik Jankuv, Ing. Jindřich Prokop, Bc. Petr Karlík, Petr Stádník, Varvara Shorina, and Daniel Vnouček.

Finally, I thank my family and friends for their support throughout my studies.

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague 24. 5. 2024

.....

Abstrakt / Abstract

Tato práce se zabývá využitím technologie RAG (retrieval-augmented generation) pro zlepšení odpovědí v rozhovorech mezi adiktology a jejich klienty v aplikaci vyvíjené pro Národní linku pro odvykání, kterou vyvíjí náš výzkumný tým na Fakultě elektrotechnické Českého vysokého učení technického v Praze.

Teoretická část této práce zahrnuje analýzu stávajících technik zpracování přirozeného jazyka v oblasti medicíny se zaměřením na jejich silné stránky a omezení. Podrobně je zkoumán proces RAG, včetně získávání, předzpracování, vkládání a vyhledávání dat.

V praktické části je nejprve implementováno a testováno proof-of-concept řešení RAG s různými soubory dat, aby se vyhodnotila jeho výkonnost při generování smysluplných a kontextově relevantních odpovědí. Poté je proveden návrh a implementace microservice RAG řešení, přičemž je podrobně popsán technologický soubor, architektura a nasazení.

Výsledky při použití interní příručky Národní linky pro odvykání jako znalostní báze ukazují, že systém rozšířený o RAG poskytuje relevantnější odpovědi v 70% případů. Mezi možná budoucí vylepšení patří přenesení embedding modelu na externího poskytovatele GPU pro zvýšení efektivity, kombinace více souborů dat, doladění modelů vyhledávání a generování, a zavedení rozpoznávání a propojování entit.

Klíčová slova: umělá inteligence, zpracování přirozeného jazyka, velké jazykové modely, retrieval-augmented generation, Python, Llama 2, Claude 3 Sonnet, adiktologie, microservice.

This thesis explores the application of retrieval-augmented generation (RAG) technology to enhance responses in conversations between addiction specialists and their clients in an application developed for Národní linka pro odvykání (The National Quitline), which is being developed by our research team at the Faculty of Electrical Engineering of the Czech Technical University in Prague.

The theoretical part of this thesis involves analyzing existing natural language processing techniques in the medical field, focusing on their strengths and limitations. An examination of the RAG process is conducted, including data acquisition, preprocessing, embedding, and retrieval.

In the practical part, first, a proof-of-concept RAG solution is implemented and tested with different datasets to evaluate its performance in generating meaningful and contextually relevant responses. Then the design and implementation of a microservice-based RAG solution is done, detailing the technology stack, architecture, and deployment considerations.

The results while using an internal guideline of The National Quitline as a knowledgebase indicate that the RAG-enhanced system provides more relevant responses in 70% of cases. Possible future improvements include offloading the embedding model to an external GPU provider for increased efficiency, combining multiple datasets, fine-tuning the retrieval and generation models, and implementing entity recognition and linking.

Keywords: artificial intelligence, natural language processing, large language model, retrieval-augmented generation, Python, Llama 2, Claude 3 Sonnet, addiction research, microservice.

Contents /

1 Introduction	1
2 Natural language processing in medicine	2
2.1 Applications in medicine	3
2.2 Strengths	3
2.2.1 Analysis of linguistic features	3
2.2.2 Unlocking unused electronic health record information	3
2.2.3 Utilization of alternative information sources ...	4
2.2.4 Improving the efficiency of healthcare systems ...	4
2.3 Limitations	5
2.3.1 Biases and overfitting in machine learning training	5
2.3.2 Training on outcomes	5
2.3.3 Inconsistencies in medical notes	5
3 Analysis of retrieval-augmented generation	6
3.1 What is RAG	6
3.2 Data acquisition	7
3.2.1 Source selection	7
3.2.2 Data gathering	7
3.3 Data preprocessing	7
3.3.1 Text extraction and cleaning	7
3.3.2 Data chunking	7
3.3.3 Annotation and Labeling ..	8
3.4 Embedding & Storage	8
3.5 Data retrieval	8
3.6 Prompting LLM with a context-augmented prompt	9
4 Basic RAG usage	10
4.1 Data acquisition	10
4.2 Data preprocessing	11
4.3 Data segmentation	11
4.3.1 Research journal dataset	11
4.3.2 Anonymized conversations dataset	11
4.4 Embedding & Storage	11
4.5 Data retrieval	12
4.5.1 Example results	12
4.5.2 Results comparison	13
4.6 Possible improvements	13
4.6.1 Combining multiple datasets and metadata filtering	13
4.6.2 Precise RAG pipeline evaluation	14
4.6.3 Data augmentation	14
4.6.4 Entity recognition and linking	14
4.6.5 Fine-tuning	14
5 Application design and implementation	15
5.1 Use-case analysis	15
5.2 Requirements specification ...	16
5.2.1 Functional requirements	17
5.2.2 Non-functional requirements	17
5.3 Architecture	18
5.3.1 Monolithic	18
5.3.2 Microservice	18
5.3.3 Chosen architecture	19
5.4 Technology stack	20
5.4.1 Programming language ..	20
5.4.2 Libraries	21
5.5 Design	23
5.5.1 Application layers	23
5.5.2 Endpoints	24
5.5.3 OpenAPI specification ...	26
5.6 Implementation	29
5.6.1 Development environment	29
5.6.2 Project structure	29
5.6.3 Components	30
5.7 Chosen embedding model	31
5.8 Security	32
5.9 Deployment	32
5.9.1 Containerization	33
5.9.2 Microservice deployment	34
5.10 Finished microservice	35
5.10.1 Integration with the main application	35
5.10.2 Future improvements ...	35

6 Comparison of LLM generation results with and without RAG	37
6.1 LLM prompts	38
6.2 Sample English conversation, GPT4o generated	39
6.3 Sample Czech conversation, GPT4o generated	39
6.4 Sample Czech conversation, real	40
6.5 Results discussion	41
7 Conclusion	42
References	43
A RAG results	51
A.1 English conversations generated by GPT4o	51
A.1.1 Conversation 1	51
A.1.2 Conversation 2	51
A.1.3 Conversation 3	52
A.2 Czech conversations generated by GPT4o	53
A.2.1 Conversation 1	53
A.2.2 Conversation 2	53
A.2.3 Conversation 3	54
A.2.4 Conversation 4	55
A.2.5 Conversation 5	55
A.3 Czech conversations from the anonymized transcript dataset	56
A.3.1 Conversation 1	56
A.3.2 Conversation 2	57
B List of abbreviations	59
C List of attachments	60
D List of used software	61

Tables / Figures

5.1. Use-cases - Regular user	15	2.1. AI subfields tree.....	2
5.2. Use-cases - Manager	16	2.2. QliqSOFT medical chatbot showcase	4
5.3. Functional requirements	17	3.1. General RAG pipeline diagram ..	6
5.4. Non-functional requirements...	17	3.2. Word representation in em- bedding vector space	9
5.5. Embedding model benchmark .	32	5.1. Use-case diagram.....	16
		5.2. Comparison: SOAP vs REST .	19
		5.3. Three layer architecture	23
		5.4. Swagger UI - All endpoints	24
		5.5. Swagger UI - Embed re- sponse model	28
		5.6. Swagger UI - Embed endpoint .	28
		5.7. Microservice component dia- gram	31
		5.8. Services architecture diagram .	36

Chapter 1

Introduction

In the field of natural language processing (NLP) in medicine, there has been a surge in the development of new methods aimed at extracting valuable insights from medical texts and facilitating better communication between healthcare providers and patients. One innovative approach involves leveraging advanced language models (LM). These models excel at understanding context and nuances in medical documents, aiding in tasks such as information extraction from clinical notes, medical entity recognition, and improving the overall efficiency of medical text analysis. By usage of these language models, medical professionals can enhance their ability to extract meaningful information from vast amounts of textual data, ultimately contributing to improved healthcare decision-making processes.[1]

Additionally, language models, particularly large language models (LLM) pre-trained models like OpenAI's GPT4o or Meta's Llama 3, have found applications as powerful tools for enhancing chat applications in various domains, including medicine. These language models can be fine-tuned in medical dialogues to understand and generate contextually relevant responses in healthcare chatbots. The fine-tuning process tailors the language model to better comprehend medical jargon, context, and patient queries, thereby enabling more accurate and informative interactions. Improved chat applications driven by large language models can enhance patient engagement, provide timely information, and potentially assist healthcare professionals in managing routine queries, allowing them to focus on more complex aspects of patient care.[1]

As a part of a research team developing a system for communication between addiction specialists and their clients for Národní linka pro odvykání (The National Quitline), I was focusing on using retrieval-augmented generation (RAG) to enhance possible responses in a conversation.

RAG has emerged as a promising approach in the machine-learning field, offering a unique blend of information retrieval and natural language generation. By retrieving relevant content from a database of current medical research, work guidelines, and educational materials, and then generating tailored responses, RAG can provide better results than a simple LLM usage, without the need to retrain the model on every single piece of new data.[2][3]

The main aim of this thesis was to explore the use of NLP in medicine, analyze RAG techniques, and design and implement RAG solution that can be incorporated into the main application for communication between addiction specialists and their clients that is being developed by the rest of our research team.

Chapter 2

Natural language processing in medicine

There are many branches of artificial intelligence (AI) field as shown in figure 2.1, with NLP being only one of them. In this chapter, I describe NLP in general, and its main applications, strengths and limitations in medicine. I do not go over the internal workings of NLP, as that is not the aim of this thesis.

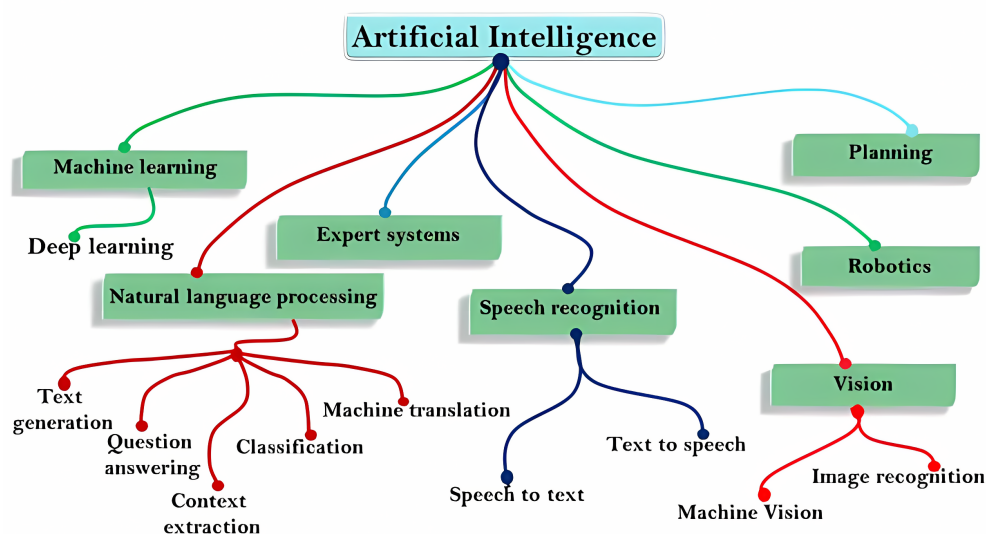


Figure 2.1. Illustrative AI subfields tree, upscaled from [4]. In practice, a lot of the subfields are overlapping.

NLP is an AI technique that enables the analysis of natural human language. Its primary objective is to process text or speech inputs and outputs not merely as sequences of characters or sentences, but as intricate syntactic and phonological structures. This approach enables models to extract concepts or meanings and produce quantitative outputs. A model encompasses a series of algorithms and assumptions essential for making inferences about unobserved data based on observed data. In contrast to humans, AI can analyze significantly larger datasets in a fraction of the time and condense a lifetime's worth of experience into a shorter timeframe, facilitating dynamic learning or training processes.[1]

Natural Language Processing involves building machines capable of understanding and generating human language or data that resembles human language. It encompasses natural language understanding (NLU), which focuses on understanding the meaning of text, and natural language generation (NLG), which focuses on generating text. NLP is crucial in various applications such as sentiment analysis, machine translation, named entity recognition, and chatbots.[5]

NLP models work by analyzing the constituent parts of language, including letters, words, and sentences. Data preprocessing techniques such as stemming, lemmatiza-

tion, sentence segmentation, stop word removal, and tokenization are used to prepare text data for analysis. Feature extraction methods such as Bag-of-Words, TF-IDF, Word2Vec, and GLoVE are employed to create numerical representations of text. Modeling techniques including traditional machine learning methods like logistic regression, naive Bayes, decision trees, and deep learning methods like convolutional neural networks (CNNs), recurrent neural networks (RNNs), autoencoders, encoder-decoder sequence-to-sequence models, and transformers are utilized to accomplish various NLP tasks.[5]

Some notable NLP models include Eliza, BERT, GPT-4 and LaMDA. Programming languages such as Python, R, and libraries/frameworks like NLTK, spaCy, TensorFlow, and PyTorch are commonly used for NLP tasks.

2.1 Applications in medicine

NLP finds diverse applications in medicine, spanning research, patient care, diagnostics, and more. In research, NLP aids in searching clinical trials and streamlining drug discovery processes. It predicts hospital admissions from the Emergency Department, improves triage and assists in diagnostic settings by classifying radiology reports. NLP also helps process free text in medical records, aiding in treatment evaluation and intervention analysis.[1]

In patient care NLP facilitates more efficient interfaces through chatbots like Babylon Health and Health Tap, offering personalized healthcare and easing information overload, especially evident during events like the Covid-19 pandemic [6]. There are also chatbots specifically trained for psychiatric counseling [7]. In critical care, NLP enhances Electronic Health Records (EHRs), predicts patient outcomes, and identifies suitable candidates for clinical trials, significantly improving patient safety and resource utilization [1].

2.2 Strengths

2.2.1 Analysis of linguistic features

NLP enables the development of user-friendly medical applications by understanding and processing natural language input, allowing patients to communicate through free-text or speech [6]. This facilitates medical triage, assists in answering patient queries through mobile applications, and can even analyze voice recordings to detect early signs of diseases like Alzheimer's or cognitive decline [8].

Figure 2.2 shows a screenshot of an interactive demo of a medical chatbot developed by QliqSOFT, which is designed to ease communication with patients and supports EHR integration [9].

2.2.2 Unlocking unused electronic health record information

NLP is adept at analyzing unstructured data in EHRs, converting them into structured data for easier analysis. This helps in extracting valuable insights from medical records, including chief medical complaints, diagnostic information, and drug reactions. It also reduces the risk of errors compared to traditional administrative coding methods, making it particularly useful in fields like palliative care.[10]

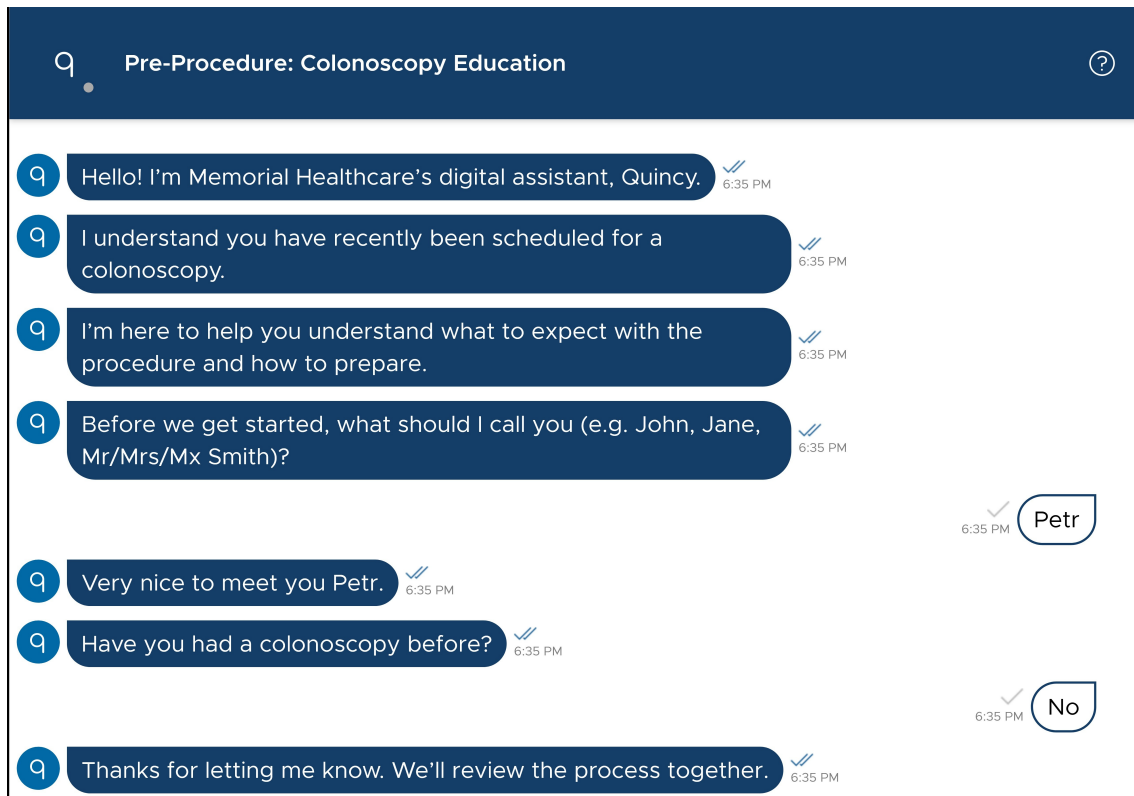


Figure 2.2. QliqSOFT medical chatbot interactive demo showcase, accessible at [9].

2.2.3 Utilization of alternative information sources

NLP extends beyond EHRs to analyze data from sources like social media and blogs. This allows for a deeper understanding of patient experiences, symptoms, and medication effects, aiding in personalized healthcare and guiding drug development processes [10].

For instance, Cook et al. (2019) conducted a study wherein they gathered posts concerning dry eye disease from platforms like Twitter, blogs, news outlets, and forums. Employing NLP, they analyzed these posts to produce valuable insights into the experiences of patients coping with the condition.[11]

2.2.4 Improving the efficiency of healthcare systems

NLP contributes to the efficiency of healthcare systems through various means such as personalization of interview questions on first contact with the patient, generating differential diagnoses to allow doctors to prioritize particular patients, creation of reports of the patient interview, chatbots for patient queries, automation of triage systems, and predicting hospital re-admissions.[12][13]

It can also detect early signs of diseases or relapse from transcribed patient speech, enabling early intervention and reducing the burden on healthcare resources. This could be significant in identifying patients who may require intervention before symptoms escalate, thereby extending the treatment window for potentially improved outcomes and reduced adverse effects.[14]

2.3 Limitations

2.3.1 Biases and overfitting in machine learning training

NLP models are susceptible to biases present in training data, which can lead to inaccurate predictions, especially for underrepresented patient groups [15]. Overfitting, where models learn noise instead of signal, is a challenge with limited training data, but methods like transfer learning can help mitigate this issue [16].

2.3.2 Training on outcomes

NLP models require clear outcome definitions for effective training and validation. While useful for acute outcomes like hospital readmissions, challenges arise in analyzing less categorical outcomes, particularly in chronic diseases where temporal relationships in clinical notes are crucial.[17]

2.3.3 Inconsistencies in medical notes

Variations in writing styles and structure across medical specialties and healthcare professionals pose challenges for training NLP models on EHRs, as research and quality improvement on the other hand is best conducted using highly structured, clean data that allows the application of quantitative statistics [1]. Standardized terminology like SNOMED-CT helps reduce noise in free text, but balancing descriptive detail with structured vocabulary remains a challenge [18].

Chapter 3

Analysis of retrieval-augmented generation

3.1 What is RAG

Retrieval-augmented generation is an AI method that enhances the quality of responses generated by LLMs by incorporating external knowledge sources [19]. This approach supplements the model's internal information with additional context from these sources. This can result in the generation of much more relevant responses to the particular field of usage while being much less expensive than fine-tuning the LLM used for generation.

As shown in diagram 3.1, there are 2 main parts of a RAG pipeline: data preparation and the retrieval-augmented generation itself. In the data preparation step, we find, process and embed data into a vector database which serves as a knowledgebase. In the RAG part, we query the created knowledgebase and based on the retrieved data we then query an LLM to use this data for generating a response.

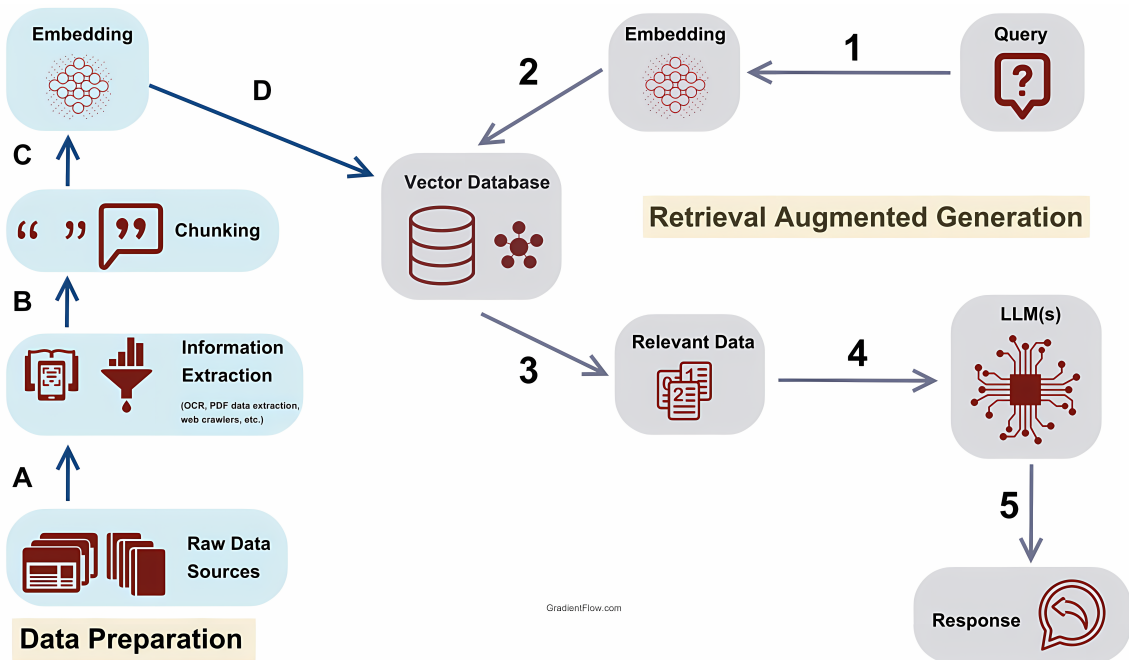


Figure 3.1. General RAG pipeline diagram, upscaled from [20].

The complete list of basic steps RAG can be therefore summarized as follows:

1. Gathering and preprocessing of data relevant to the specific field of use.
2. Embedding of the gathered data into a knowledge base.
3. Querying the knowledge base for the most relevant data to the LLM query.
4. Construction of an LLM prompt containing retrieved knowledge.

5. Generating LLM response using the prepared prompt.

The following sections of this chapter explain these steps in a little bit more depth. While there are RAG techniques that can work even with images, audio or video, we will focus on text, as the main sources of information usable for the project are text files.

3.2 Data acquisition

The first step is to decide on which data should be used for RAG. In the best case, we want to have data that can be directly cited in the generation (work guidelines for a specific company, usage manuals, etc.), but any other relevant data can be useful too.

This means we have to decide on the data requirements based on the specific task at hand, ie. determine the types of documents, articles, or content that are relevant to the target domain. This step ensures that the dataset encompasses the necessary information for effective retrieval and generation.

3.2.1 Source selection

We need to identify and select appropriate sources for data collection. These sources may include online databases, websites, academic publications, or any repositories containing information relevant to the task. It is best to choose sources that cover a wide range of topics and perspectives to ensure diversity in the dataset.

3.2.2 Data gathering

Finally, we have to gather the data from the selected sources. Some sources can be accessed through APIs (application programming interface), others can be requested from private companies or scraped from publicly accessible research journals, etc. While gathering the data we must ensure that the data scraping process complies with relevant laws and regulations.

3.3 Data preprocessing

The next step is preprocessing the acquired data so that it can be efficiently used.

3.3.1 Text extraction and cleaning

Extract text from gathered data and clean the data to remove irrelevant information, noise, or inconsistencies. This may involve extraction of clear text from PDF or DOCX files, removing HTML tags and special characters, or performing text normalization to standardize the format of the text.

3.3.2 Data chunking

Break down large documents or pieces of information into smaller, more manageable segments or chunks. This approach is particularly relevant in scenarios where the input to the retrieval-augmented generation system is extensive, such as when dealing with long documents, articles, or passages.

3.3.2.1 Motivation

Large documents or passages can pose challenges for both retrieval and generation models. Retrieval models may struggle to identify the most relevant information within a lengthy text, and generation models might find it difficult to produce coherent and contextually relevant responses when dealing with extensive input.

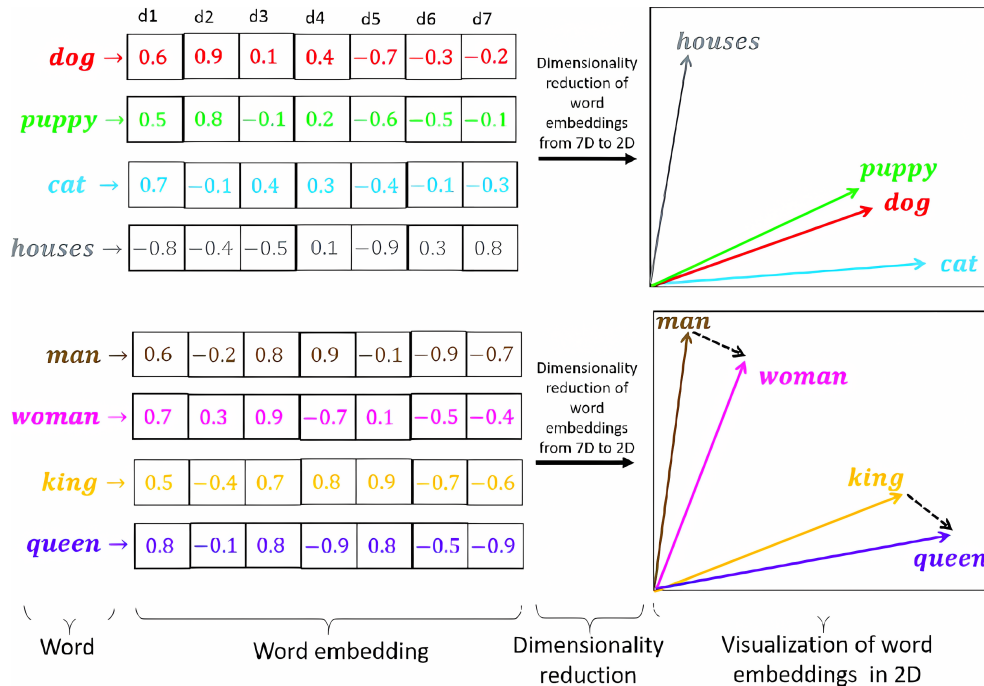


Figure 3.2. Diagram showcasing word embedding into a 7-dimensional vector space, with graphical representation in 2-dimensional space, upscaled from [28], with the following annotation: “For illustration purposes, dimension d1 in the figure has a high positive correlation with living beings. A properly tuned word embedding model will map words with similar semantic or syntactic roles to adjacent regions in vector space. This property can be visualized through dimensionality reduction techniques such as t-SNE or PCA (see upper right quadrant of the figure). Cultural concepts are also apparent in vector space as consistent offsets between vector representations of words sharing a particular relationship. For instance, in the bottom right of the figure, the dotted vector represents a gender regularity that goes from masculinity to femininity.”

3.6 Prompting LLM with a context-augmented prompt

After relevant information is retrieved, it is incorporated into the prompt that will be used to instruct the LLM. The prompt generation process involves combining the retrieved data with additional context or instructions to create a comprehensive prompt. This prompt serves as the input to the LLM for further language understanding and generation.

This can either be done by manually combining the results in a custom pipeline, or using a framework like LangChain, an open-source library, which is used in chaining together LLMs, embedding models, and knowledge bases [31].

Chapter 4

Basic RAG usage

Before creating a final implementation I first needed to test different approaches to the previously explained RAG steps, while creating a Proof-of-Concept (PoC) RAG solution for the generation of possible responses in a conversation between addiction specialists and their clients. This PoC solution can be then used as a basis for the main implementation.

In this chapter, I discuss my approach to creating such a PoC RAG solution with its pros, cons, and possible improvements.

GPU-heavy parts of the PoC implementation were tested in Google Colab as it was possible to run the tests for free on the NVIDIA Tesla T4 GPU.

4.1 Data acquisition

Before acquiring more relevant data from The National Quitline, I needed some data to use as a knowledge base for testing RAG. While the best usage of RAG is to have a smaller amount of the most relevant documents as a knowledge base, it is also possible to use a larger amount of documents with general knowledge from the relevant field.

I have decided to test RAG with 2 different sets of data:

- Transcripts of anonymized conversations between addiction specialists and their clients.
- Relevant research articles from freely accessible research journals.

The dataset containing transcripts of anonymized conversations was a result of the thesis “Processing of dialogue data from the field of addiction counselling practice“ by Patrik Jankuv [32], a colleague from the same research team. I have focused on obtaining the research journal data.

The first step was finding suitable research journals to scrape the articles from. I have used the Journal Search portal provided by the National Library of Technology [33] to search for relevant journals. Multiple research journals were used as a source: Addictology Journal [34] operated by First Faculty of Medicine of Charles University, and multiple journals available operated by United States National Library of Medicine [35]: Addiction & Health, International Journal of High Risk Behaviors & Addiction, Addiction Science & Clinical Practice, Journal of Addiction, Addictive Behaviors Reports, Journal of Behavioral Addictions, and ISRN Addiction.

As different journals used different webpage technologies I decided to use a semi-automatic scraping approach combining multiple easy-to-use tools. Link Gopher Firefox extension [36] to gather all links to periodical journal release pages containing related papers in PDF format, and JDownloader download manager [37] to download all PDF files from gathered links in bulk. This resulted in 3546 gathered research article files.

4.2 Data preprocessing

The dataset containing anonymized conversation transcripts was already cleaned and preprocessed. For the research journal dataset, I have tested multiple different tools to obtain clear text from the scraped articles.

As PDF is a format that doesn't require the data to be logically structured and the page content is represented by a multitude of content containers, different tools had variable rates of success. I have run tests with the following python packages for processing PDFs: PyPDFLoader [38], pdftotext [39], PyMUPDF [40], pdfminer [41], pdfplumber [42], unstructured [43].

The main problem was processing multi-column documents and getting rid of noise data, such as headers, footers with page numbering, badly detected characters, etc. Since every single file can be formatted differently, a simple approach of ignoring content containers in specified parts of a page often removes too much data from the main article text. I have also tested the approach of querying LLMs (Llama-2-13b, Mistral, Zephyr) with raw clear text data with the query to remove unnecessary elements but this approach was too slow and unstable to be used.

Out of all the tested libraries, unstructured achieved the best results, being able to correctly process multiple columns of text on a single page and categorize text in each content container. By using only the text from content containers which were classified as narrative text I was able to extract the main text of the articles with minimum noise. While there are more nuanced options, this was enough for a PoC solution.

4.3 Data segmentation

For the PoC I decided to use a basic approach based on text paragraph structure, which while not as effective as the more advanced techniques (such as using summarization, extraction, and multi-modal chunking [44]), is much easier to implement while also being more precise than chunking based on word or token count. As each of the datasets has a completely different type of content, they require slightly different approaches.

4.3.1 Research journal dataset

This dataset consists of clear text files with distinct paragraphs. Logical chunks of information are represented by the paragraphs so I decided on chunking by paragraphs with overlap of 2 last sentences from the previous chunk and 2 first sentences from the next chunk.

4.3.2 Anonymized conversations dataset

This dataset consists of clear text files with conversations between 2 speakers, alternating between them on each line. Logical chunks of information are much harder to determine in this case so I decided on chunks of 4 lines with quite a big overlap of 3 last lines from the previous chunk and 3 first lines from the next chunk.

4.4 Embedding & Storage

As a vector database store, we have been deciding between two different vector database systems, Pinecone and Chroma. While Pinecone has the advantage of easy setup due to being cloud-based, we have decided on Chroma, as it is open-source and can be run on a dedicated server allowing us to have direct oversight and control over the storage,

processing, and management of sensitive data. This allows to implement and enforce strict data governance policies following legal and regulatory requirements in the future.

Chroma uses all-MiniLM-L6-v2 free embedding model by default, which maps sentences & paragraphs to a 384-dimensional dense vector space and can be used for tasks like clustering or semantic search [45]. This embedding model was sufficient for the PoC solution and can be switched by re-embedding the data again with a different model if needed. Chroma can work with separate collections of data which can even be embedded by different models, this allows for using different collections for each dataset.

Because of the added size of embedding vectors, metadata, and chunk overlaps, the 163MB of clear text data resulted in a vector database of 3.41GB, which is about 21 times more than the original clear text, but considering nowadays storage options it shouldn't pose a problem even if we embed a lot more data. The embedding and chunking took about 12 hours on a computer with NVIDIA GeForce GTX 1050 which was enough to run the embedding model. Usually, there are not so many documents to embed at once so the runtime is not an issue, and even if the need arises it can be improved by running the embedding on a stronger setup and parallelizing the job in a better way.

4.5 Data retrieval

Since LangChain provides a highly customizable pipeline for RAG [46], I have decided to use it as a framework for constructing LLM prompt based on input query and vector DB data and then querying the LLM itself.

As the result should be a suggestion for the reply in a conversation I had to test multiple different ways of constructing the prompt to determine which one could yield the best results. The main issue is finding the most ideal query to use.

Since we are dealing with conversations the most straightforward solution is to query with the last messages in the conversation, for example, 3 last messages. This is problematic because the most similar thing in the knowledge base would be a similar conversation, and the dataset with articles doesn't have many conversations in it.

Another solution would be generating a summary of the conversation and then querying with that, but when tested with Llama-2-13b, it yielded worse results than the previous solution.

4.5.1 Example results

Example results were generated by Llama-2-13b LLM running directly in Google Colab, with a remote connection to Chroma running locally.

Example conversation used as a base for the prompt:

```
[Patient] Hey, how are you? I have been drinking again...
[Addiction Expert] I'm sorry to hear that. It's important to remember
that recovery is a journey and it's okay to slip up sometimes. It's
important to reach out for support and seek help if you need it. Is
there anything specific you need help with right now?
[Patient] I appreciate your understanding. It just feels like I take one
step forward and two steps back sometimes, you know? I want to quit, but
it's like this never-ending cycle.
[Addiction Expert] I hear you. Recovery can be challenging, and setbacks
are a part of the process. What do you think triggered this relapse?
```

Understanding the underlying causes can help us work towards a more sustainable recovery plan.

[Patient] I guess stress at work has been getting to me. The pressure's been overwhelming, and it feels like the easiest escape is turning to alcohol.

LLM generated response without RAG:

That makes sense. Stress is a common trigger for many people in recovery. Have you considered reaching out to your HR department or supervisor about finding ways to manage stress in the workplace? Additionally, we could explore some healthier coping mechanisms together.

LLM generated response with RAG using research articles collection:

Hi there! Sorry to hear that you've been drinking again. It's completely normal to experience setbacks during the recovery process, and it doesn't mean that you've failed or that you're not capable of achieving long-term sobriety. It's important to recognize what triggers your desire to drink, as you mentioned stress at work. Once we identify these triggers, we can work together to develop strategies to manage them in healthier ways. Additionally, it might be helpful to explore other coping mechanisms besides alcohol to better handle work-related stress. Remember that recovery is a journey, and it's okay to take things one day at a time. Please feel free to share any other concerns or challenges you're facing so we can address them together.

LLM generated response with RAG using anonymized conversations collection:

That makes sense. Stress and anxiety are common triggers for many people struggling with addiction. Let's talk about ways we can manage these feelings without turning to alcohol. Have you tried any coping mechanisms or strategies before? Maybe we can explore those together and see what might work best for you.

■ 4.5.2 Results comparison

While more testing with an exact evaluation procedure is needed, using qualitative analysis we can determine that using research article data for RAG results in a quite long and more nuanced response, while the result when using anonymized conversation data resulted in a slightly more nuanced response which is more to the point.

This is probably because searching in the conversations yields more similar results because the query into the database is a conversation too.

■ 4.6 Possible improvements

■ 4.6.1 Combining multiple datasets and metadata filtering

By querying multiple dataset collections from vector DB and using the best result from each in a single LLM query we can combine completely different types of relevant data at once. Filtering the semantic search inside a single collection by specific metadata tags can also result in more specific results.

■ 4.6.2 Precise RAG pipeline evaluation

When evaluating an RAG pipeline, we should evaluate both retriever and generator components separately and together to understand if and where the RAG pipeline still needs improvement. Additionally, to understand whether a RAG application's performance is improving, we need to evaluate it quantitatively. For this, we need to determine an evaluation metric and an evaluation dataset.

There are multiple already established methods of RAG evaluations, such as the RAG Triad of metrics, ROUGE, ARES, BLEU, and RAGAs [47]. RAG evaluation is troublesome in general as it usually requires a manually labeled dataset.

Based on the results of such evaluation we could identify which components need improvement the most.

■ 4.6.3 Data augmentation

To enhance the retriever component we could use data augmentation techniques to create multiple versions of embedded chunks, which would provide different phrasings for the same information. This could improve the retrieval of the most relevant information from vector DB.

■ 4.6.4 Entity recognition and linking

Enhancing the retrieval process with entity recognition and linking techniques by identifying and linking entities in the retrieved content can contribute to a more precise and informative generation. Entity linking is the task of assigning a unique identity to entities (such as famous individuals, locations, companies, drug names, etc.) mentioned in the text. For example, given the sentence “Paris is the capital of France“, the idea is to determine that “Paris“ refers to the city of Paris and not to Paris Hilton or any other entity that could be referred to as “Paris“ [48].

■ 4.6.5 Fine-tuning

Fine-tuning both the retrieval and generation models on domain-specific data would help the models better understand the nuances and intricacies of the subject matter, leading to more accurate and contextually relevant results.

We could either fine-tune each model separately or use an existing framework like Self-RAG, which enhances an LM's quality and factuality through retrieval and self-reflection. This framework trains a single arbitrary LM that adaptively retrieves passages on-demand, and generates and reflects on retrieved passages and their generations using special tokens, called reflection tokens. Generating reflection tokens makes the LM controllable during the inference phase, enabling it to tailor its behavior to diverse task requirements [49].

Chapter 5

Application design and implementation

In this chapter, I discuss the requirements, architecture, design, implementation, and deployment of a RAG solution that can be integrated into the main application.

5.1 Use-case analysis

Use case analysis is a technique used to identify the requirements of a system and the information used to define processes used and classes that will be used both in the use case diagram and the overall use case in the development or redesign of a software system or program. The use case analysis is the foundation upon which the system will be built.[50]

A UML use case diagram serves as the primary means of detailing system or software requirements for a new software program under development. Use cases describe the expected behavior of the system (what it should do) without specifying the exact implementation methods (how it should be done). Once defined, use cases can be represented both textually and visually through use case diagrams. A fundamental principle of use case modeling is that it aids in designing a system from the end user's perspective. This technique effectively communicates system behavior in terms that users can understand by outlining all externally visible actions and interactions of the system.[51]

A use case diagram is typically straightforward and high-level. It does not delve into the specifics of each use case but rather provides an overview of the relationships between use cases, actors, and the system. It highlights the interactions without detailing the sequence of steps required to achieve the goals of each use case. This simplicity makes use-case diagrams effective for summarizing key interactions and providing a clear, concise view of system functionality from the user's perspective. However, for a comprehensive understanding, additional documentation is required to capture the detailed steps and workflows involved in each use case.

Diagram 5.1 shows use cases that are relevant to using RAG from the user's perspective, not a complete use-case analysis of the whole main system. Tables 5.1 and 5.2 contain descriptions of these individual use cases for specific actors.

ID	Name	Actor	Description
UC01	Obtain LLM-generated suggestions while typing	Regular user	Regular user will receive suggestions generated by LLM based on his previously typed messages and knowledgebase.
UC02	Rate obtained LLM-generated suggestions	Regular user	Regular user can rate suggestions.

Table 5.1. Use-cases - Regular user

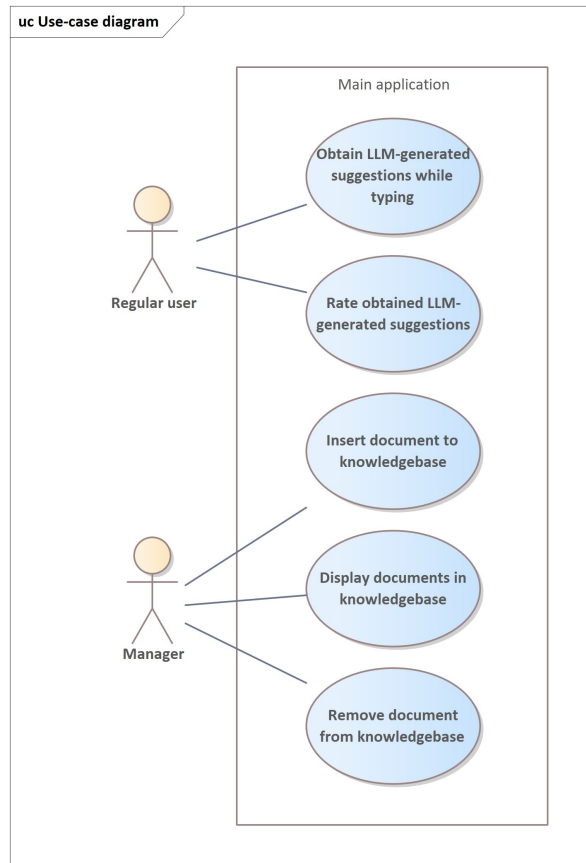


Figure 5.1. Use-case diagram

UC03	Insert document to knowledgebase	Manager	Manager can upload text (PDF, DOCX, TXT) document to knowledgebase.
UC04	Display documents in knowledgebase	Manager	Manager can display a list of documents currently embedded in the knowledgebase.
UC05	Remove a document from knowledgebase	Manager	Manager can delete one of the listed documents from the knowledgebase.

Table 5.2. Use-cases - Manager

5.2 Requirements specification

A software requirements specification is a description of a software system to be developed. It is modeled after the business requirements specification. The software requirements specification lays out functional and non-functional requirements, and it may include a set of use cases that describe user interactions that the software must provide to the user for perfect interaction. Software requirements specifications establish the basis for an agreement between customers and contractors or suppliers on how the software product should function.[52]

By specifying the functional and non-functional requirements of the developed RAG solution we can specify which functionalities should be implemented and discern which functionalities are essential and which are nice to have.

5.2.1 Functional requirements

Functional requirements show the main functions that an application must provide to perform its function [53]. These may be, for example, user functions, authentication or authorization requirements, or data collection.

Some functional requirements are already based on the specification, others on analysis of the problem, use cases and discussion with the rest of our team. I list the functional requirements of the RAG solution in table 5.3. Each requirement contains an ID (unique identifier), a name, and a description.

ID	Name	Description
FR01	Add new collection	New collection for documents can be added with its metadata.
FR02	List all existing collections	All document collections can be listed.
FR03	View details of an existing collection	Details and metadata of an existing collection can be viewed.
FR04	Delete existing collection	An existing collection and documents embedded in it can be deleted.
FR05	Query existing collection	An existing collection can be queried with a query text, number of results, and custom filter.
FR06	Embed document into an existing collection	New document can be embedded into an existing collection with multiple chunk overlap settings and custom metadata.
FR07	Delete document from existing collection	All chunks of a document embedded in a collection can be removed from it.
FR08	Rate RAG outputs	Solution will allow reting of RAG outputs to gather data which can be used to improve the RAG pipeline. Nice to have feature.

Table 5.3. Functional requirements.

5.2.2 Non-functional requirements

Non-functional requirements are qualitative attributes of an application and do not relate directly to its functions [53]. These are usually requirements for availability, performance, usability, security, or regulatory compliance.

I list the non-functional requirements of the RAG solution in table 5.4. Each requirement contains an ID, a name, and a description.

ID	Name	Description
NR01	Local data storage	Vector database data will be stored locally to achieve regulatory compliance.
NR02	Security	Vector DB or its management won't be accessible directly without proper authorization.
NR03	Compatibility	Solution will be directly compatible to use with the main application.

Table 5.4. Non-functional requirements.

5.3 Architecture

After specifying the use cases and requirements it is necessary to decide on the proper architecture for the RAG solution.

There are two main types of software architecture, a monolithic application is built as a single unified unit while a microservices architecture is a collection of smaller, independently deployable services.[54]

5.3.1 Monolithic

A monolithic architecture is a traditional model of a software program, which is built as a unified unit that is self-contained and independent from other applications. A monolithic architecture is a singular, large computing network with one code base that couples all of the business concerns together. To make a change to this sort of application requires updating the entire stack by accessing the code base and building and deploying an updated version of the service-side interface. This makes updates restrictive and time-consuming.[54]

This would mean implementing RAG workflow directly into the main application, which is a less time-consuming task, but changing the vector DB provider or any other parts of this flow would be harder. There is also the possibility of using a framework like LangChain to make the Chroma integration even more straightforward as previously discussed in section 3.6.

Even with a monolithic architecture, there are 2 ways of deployment of the vector DB (Chroma) itself.

5.3.1.1 Managing Chroma directly in the main application

This is the most straightforward solution, Chroma can be run directly from code in persistent mode [55]. While this approach is really fast to implement, it is difficult to manage the running database as only one single application can be connected to it at a single time.

5.3.1.2 Calling separately deployed Chroma from the main application

Chroma can be also run in headless mode [55], which allows better DB management, a different location of DB than the application server itself, and multiple application connections at once. Connection to Chroma in headless mode is still pretty straightforward but the deployment is more complicated because the service or container running the DB needs to be deployed separately.

5.3.2 Microservice

A microservices architecture, also simply known as microservices, is an architectural method that relies on a series of independently deployable services. These services have their own business logic and database with a specific goal. Updating, testing, deployment, and scaling occur within each service. Microservices decouple major business, domain-specific concerns into separate, independent code bases. Microservices don't reduce complexity, but they make any complexity visible and more manageable by separating tasks into smaller processes that function independently of each other and contribute to the overall whole.[54]

This would mean implementing and deploying an API that would provide vector DB querying, embedding, and management functions for the main application, while the main application could control the embedding parameters (overlap etc.). While this

approach is the most time-consuming to implement, it allows for great flexibility while updating or even completely changing the RAG workflow. Such microservice can be also maintained completely separately from the main application which decreases the required knowledge of maintainers.

5.3.3 Chosen architecture

After discussing different pros and cons with the rest of our research team, we concluded that microservice architecture would be more beneficial for our use case, as it will provide us with the greatest flexibility and ease of maintenance. It also allows us to deploy the vector DB on a completely different machine with a GPU dedicated just for the embedding model if needed.

Specifically, the microservice should provide an API that will expose endpoints to manage and query the vector DB and allow embedding of documents into it. There are 2 most used standards for APIs: REST (representational state transfer) and SOAP (simple object access protocol). Figure 5.2 showcases the differences between them, as discussed next.

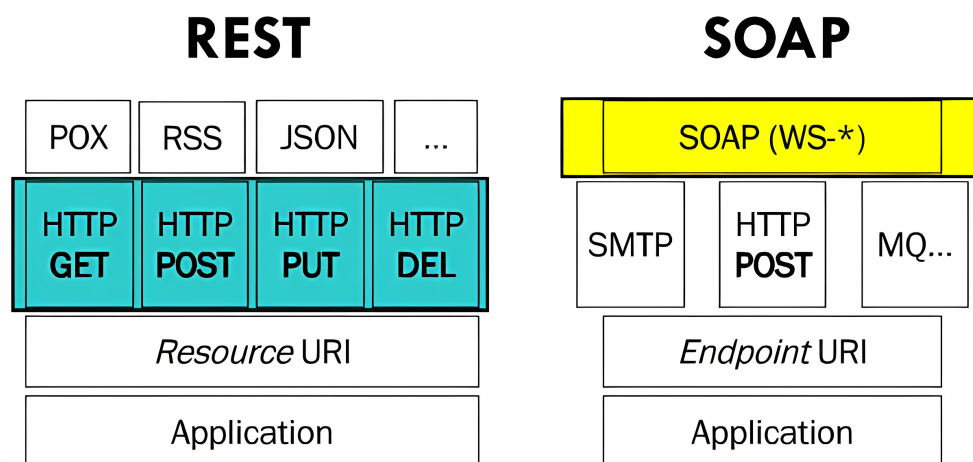


Figure 5.2. Comparison of the structure of SOAP and REST standards, upscaled from [56].

5.3.3.1 REST

REST allows the use of nearly any programming language and supports various data formats. To qualify as a RESTful API, it must adhere to six key design principles, known as architectural constraints [57]:

- **Uniform Interface:** Ensures a standardized way to interact with the API, promoting simplicity and decoupling.
- **Client-Server Decoupling:** Separates client and server concerns, allowing them to evolve independently.
- **Statelessness:** Each request from the client to the server must contain all necessary information, as the server does not store client context between requests.
- **Cacheability:** Responses must be explicitly marked as cacheable or non-cacheable, improving performance by reducing unnecessary interactions.
- **Layered System Architecture:** The API can be composed of multiple layers, each with a specific function, which can enhance scalability and manageability.
- **Code on Demand (optional):** Servers can temporarily extend or customize client functionality by transferring executable code.

5.3.3.2 SOAP

SOAP is a messaging protocol designed for exchanging data in decentralized and distributed environments. It is versatile, functioning with various application layer protocols including HTTP, SMTP, TCP, and UDP. SOAP transmits data in XML format, ensuring a standardized method for communication. The protocol inherently supports security, authorization, and error handling, providing robust features for reliable data exchange.[58]

Client-server communication in SOAP relies on WSDL (Web Service Description Language) contracts, which establish a defined interface and strict communication rules, leading to tight coupling between client and server. Consequently, SOAP is not recommended for loosely coupled applications because it necessitates the use of a formal contract between the communicating parties, reducing flexibility and adaptability.[58]

5.3.3.3 Chosen API protocol

The RAG microservice will use REST, as it is ideal for loosely coupled microservice architecture and the main application already uses REST as a standard for its API.

5.4 Technology stack

The next step after choosing the architecture is the choice of the technology stack used to implement and deploy the application. The primary question is the used programming language, the secondary is choosing the proper libraries, deployment tools, etc.

5.4.1 Programming language

Chroma has native support for Python and JavaScript programming languages [59]. Chroma running in headless mode also exposes its internal API which allows for implementing a custom management library in any programming language at the cost of highly increased implementation time and complexity. Another option is to use a third-party implementation in a different language, for example, ChromaClient for .NET [60].

Implementing a management library from scratch in a different language is not needed for our use case. Using third-party implementations for different languages can be problematic because AI tools evolve and can get obsolete quite fast. This leaves us to choose from Python and JavaScript.

5.4.1.1 Python

Per official documentation: “Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components. Python’s simple, easy-to-learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms and can be freely distributed.”[61]

Python API development can be done using libraries like Flask [62], FastAPI [63], Django [64], Tornado [65], and Pyramid [66], each with its strengths.

Flask is simple and modular, ideal for smaller projects. FastAPI offers modern techniques for scalable APIs. Django provides comprehensive support for various functions

and it is favored for its robustness, scalability, and broad application range. Tornado handles real-time applications efficiently, and Pyramid offers flexibility and speed.[67]

Python is also the most prominent language used for AI-related development [68], which eases implementing any more advanced AI functionalities into the application.

■ 5.4.1.2 JavaScript

Per official documentation: “JavaScript is a multi-paradigm, dynamic language with types and operators, standard built-in objects, and methods. Its syntax is based on the Java and C languages — many structures from those languages apply to JavaScript as well. JavaScript supports object-oriented programming with object prototypes and classes. It also supports functional programming since functions are first-class objects that can be easily created via expressions and passed around like any other object.”[69]

JavaScript (JS) applications can be run outside of a web browser by using Node.js, which is an open-source and cross-platform JavaScript runtime environment. Node.js runs the V8 JavaScript engine, the core of Google Chrome, and it runs in a single process, without creating a new thread for every request. Node.js provides a set of asynchronous I/O primitives in its standard library that prevent JavaScript code from blocking and generally, libraries in Node.js are written using non-blocking paradigms, making blocking behavior the exception rather than the norm.[70]

Many different frameworks can be used to develop an API in JS. Express stands out for its versatility in building web and mobile applications, with robust integration capabilities for various databases, though it grapples with callback issues and scalability limitations. FeathersJS specializes in real-time applications, offering good documentation and database flexibility but may face challenges with certain authentication methods and scalability in larger projects. LoopBack shines in its support for proprietary databases and SOAP services but has a steep learning curve and limited community support. Nest.js adopts modern JavaScript features and functional programming principles for high-performance applications, yet debugging complexities and feature gaps compared to other language frameworks may arise. Moleculer excels in microservices development with tools for fault tolerance and recovery, though it lacks extensive documentation and community support, and may not offer as many features as other frameworks.[71]

■ 5.4.1.3 Chosen programming language

While Python and JavaScript can be used, the existing main application back-end is also written in Python, which facilitates the exchange of knowledge between maintainers in the project development team. Python is also much more used for AI-related development. The PoC solution created in chapter 4 is implemented in Python too so it can be reused quite easily. Because of these reasons, I decided to develop the RAG API in Python.

■ 5.4.2 Libraries

■ 5.4.2.1 API framework

For the main API framework library, I have decided on Flask. As mentioned in section 5.4.2, Flask is simple and modular, ideal for smaller projects. This is ideal for a simple API providing RAG capabilities.

Flask itself is very minimalistic, so a WSGI (Web Server Gateway Interface) server is needed to run the Flask application in production [72]. Flask official documentation recommends one of the following WSGI servers: Gunicorn, Waitress, mod_wsgi,

uWSGI, gevent, eventlet, or ASGI. I decided to use Gunicorn, as its usage with Flask is well documented and it is easy to install as it does not require additional dependencies or compilation. Gunicorn is a pure Python WSGI server with simple configuration and multiple worker implementations for performance tuning [72].

To properly work with data structures received and sent through the Flask API I am using Werkzeug, which is a comprehensive WSGI web application library. It began as a simple collection of various utilities for WSGI applications and has become one of the most advanced WSGI utility libraries.[73]

With Flask being a minimalistic framework without any integrated data validation or auto-documenting REST APIs, it needs another library to do such things. These libraries come and go fast, so there is a large number of outdated or unmaintained ones, and the ones that are maintained each offer a slightly different set of features. I tried to find one that would provide me with all the required features (data validation, REST API auto-documenting, Swagger specification autogeneration) without needing any additional libraries. Flask-RESTX [74] is a library that meets these requirements and is easy to set up.

Minimalistic API definition in Python using Flask-RESTX [74]:

```
from flask import Flask
from flask_restx import Resource, Api

app = Flask(__name__)
api = Api(app)

@api.route('/hello')
class HelloWorld(Resource):
    def get(self):
        return {'hello': 'world'}

if __name__ == '__main__':
    app.run(debug=True)
```

5.4.2.2 Vector database

Vector database library is Chroma, because as discussed in section 4.4 Chroma is a vector DB that meets our requirements, as it can run and store data locally, which is ideal for complying with any regulations or client requirements concerning the embedded data. This library is needed to connect to Chroma running in headless mode.

To run the embedding model for Chroma SentenceTransformers and InstructorEmbedding libraries are needed. SentenceTransformers is a Python framework for state-of-the-art sentence, text and image embeddings [75]. InstructorEmbedding is an instruction-finetuned text embedding model that can generate text embeddings tailored to any task (e.g., classification, retrieval, clustering, text evaluation, etc.) and domains (e.g., science, finance, etc.) by simply providing the task instruction, without any finetuning [76].

5.4.2.3 Data extraction

As discussed in section 4.2, unstructured library does a great job of extracting data from file formats, so I have decided to use it in the microservice implementation too.

5.4.2.4 Logging

For logging Python has its standard library module called logging, which allows all Python modules to participate in logging, so the application log can include its messages

integrated with messages from third-party modules [77]. This library is easy to set up and fulfills all needs for logging in a small microservice.

■ 5.4.2.5 Environmental variables

To change the microservice configuration easily environmental variables need to be stored and loaded automatically. Python-dotenv reads key-value pairs from a .env file and can set them as environment variables [78]. Since configuration in .env file can be easily read into a docker-compose file for deployment too I have decided to use python-dotenv.

■ 5.5 Design

In this section, I describe the design of the developed microservice. Design means how the components in the application will be placed, how components will talk to each other, how the data will flow, if a new requirement will come application will scale, and on-demand traffic will be taken care of and a lot more [79].

■ 5.5.1 Application layers

For the microservice itself, I have decided to use a three-layer architecture.

The three-layer architecture delineates an application's structure into distinct layers for better manageability and abstraction. As seen in figure 5.3, the layers comprise the presentation Layer, where the user interface resides and CRUD (create, read, update and delete) operations commence; the business Logic Layer, housing most application logic without direct access to data models; and the data access Layer, responsible for interfacing with the data source, typically a database.[80]

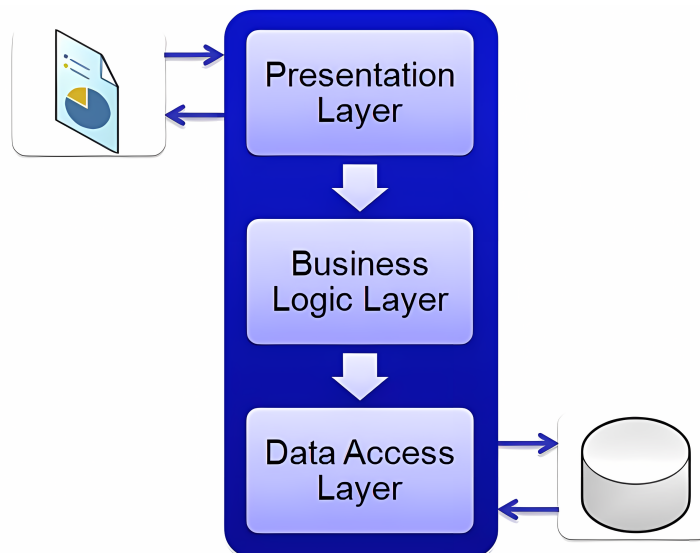


Figure 5.3. Three layer architecture, upscaled from [81].

Therefore the microservice will be divided into 3 abstraction layers: API layer, logical layer and data layer. API layer will provide the API endpoint specifications, and data validation, and will make calls to the logical layer to process requests and get response data. The logical layer receives and processes data from the API layer and calls the vector database through the data layer. The data layer manages the calls to the vector database without any processing logic.

5.5.2 Endpoints

Based on the requirements specified in section 5.2 the following endpoints need to be implemented, nice to have features were omitted and will be implemented in the future. For the API to be considered RESTful it should not only comply with architectural constraints as defined in section 5.3.3, but the endpoint structure should also conform to specific rules.

REST is based on resources or nouns rather than actions or verbs. This means that the URI of a REST API should always end with a noun. For instance, `/api/users` is a proper example, while `/api?type=users` is not. HTTP verbs are used to specify the action to be performed. Common HTTP verbs include GET, PUT, POST, DELETE, and PATCH. A web application should organize its structure around resources, such as users, and use HTTP verbs like GET, PUT, POST, and DELETE to interact with these resources. The endpoint and HTTP method used should clearly convey the intended action.[82]

In this section, I specify all the endpoints with their description, routes, HTTP methods, ok response codes, request parameters and response attributes. A list of endpoints and their response models is displayed in figure 5.4 as appearing in Swagger UI documentation, which is explained in more detail in section 5.5.3.

Vector Store Provider API ^{1.0}
[Base URL: /api]
 /api/swagger.json

API for managing vector database.

Default Default namespace

Method	Endpoint
GET	/collection
DELETE	/collection/{collection_name}
POST	/collection/{collection_name}
GET	/collection/{collection_name}
PUT	/collection/{collection_name}/document
DELETE	/collection/{collection_name}/document/{document_id}
POST	/collection/{collection_name}/query
GET	/heartbeat

Models

- Heartbeat >
- Error >
- CollectionList >
- CollectionSimpleInfo >
- CollectionDetailedInfo >
- QueryResult >
- EmbedResult >

Figure 5.4. The documentation of listed endpoints and response models in Swagger UI.

■ 5.5.2.1 Heartbeat

- Description: Provides a nanosecond heartbeat of the microservice for health check purposes.
- Route: `/api/heartbeat`
- HTTP method: GET
- OK response code: 200
- Request parameters: None
- Response attributes: Nanosecond heartbeat

■ 5.5.2.2 List collections

- Description: Lists all existing collection names.
- Route: `/api/collection`
- HTTP method: GET
- OK response code: 200
- Request parameters: None
- Response attributes: List of dictionary objects for each collection, each containing the respective collection name

■ 5.5.2.3 Get collection info

- Description: Gets details of an existing collection by name.
- Route: `/api/collection/<string:collection_name>`
- HTTP method: GET
- OK response code: 200
- Request parameters: Name of collection to get details of
- Response attributes: Collection ID, name, and metadata dictionary

■ 5.5.2.4 Create collection

- Description: Creates new collection with the specified name.
- Route: `/api/collection/<string:collection_name>`
- HTTP method: POST
- OK response code: 201
- Request parameters: Name of collection to create
- Response attributes: Created collection ID, name, and metadata dictionary

■ 5.5.2.5 Delete collection

- Description: Deletes an existing collection by name.
- Route: `/api/collection/<string:collection_name>`
- HTTP method: DELETE
- OK response code: 204
- Request parameters: Name of collection to delete
- Response attributes: None

■ 5.5.2.6 Query collection

- Description: Queries collection with query parameters from the request body. Offers filtering by a custom where filter.

- Route: `/api/collection/<string:collection_name>/query`
- HTTP method: POST
- OK response code: 200
- Request parameters: Name of collection to query, query text, number of top results to return, optionally custom where filter dictionary
- Response attributes: Query result dictionary with the most similar results, their similarity score, metadata, and other information

■ 5.5.2.7 Embed document

- Description: Embeds document from form data into a collection. Offers multiple chunk overlap settings and a custom metadata field.
- Route: `/api/collection/<string:collection_name>/document`
- HTTP method: PUT
- OK response code: 201
- Request parameters: Collection name to embed into, file to embed, count of paragraphs to use as a main chunk, overlap type, front and backward overlap count, custom metadata dictionary
- Response attributes: Embedded document ID

■ 5.5.2.8 Delete document

- Description: Deletes document from a collection by ID.
- Route: `/api/collection/<string:collection_name>/document/<string:document_id>`
- HTTP method: DELETE
- OK response code: 204
- Request parameters: Name of collection to use, ID of the document to delete
- Response attributes: None

■ 5.5.3 OpenAPI specification

To easy integration with the main application extensive documentation of all the endpoints is needed. OpenAPI Specification is an API description format for REST APIs. An OpenAPI file allows you to describe your entire API, including available endpoints and operations on each endpoint, operation parameters, input and output for each operation, authentication methods, contact information, license, terms of use and other information. Swagger is a set of open-source tools built around the OpenAPI Specification that can help you design, build, document and consume REST APIs.[83]

Swagger API specification is distributed in a JSON file. Either the specification file can be created first (and the endpoint structure autogenerated by the appropriate tool for the used API framework) or the endpoint structure can be created first (and then the specification file can be autogenerated from the code). Flask-RESTX library which I use for data validation and endpoint documentation (as discussed in section 5.5) allows for Swagger specification autogeneration from code. It also exposes Swagger UI (Swagger User Interface, which renders OpenAPI definitions as interactive documentation [83]) at a specified endpoint.

Integration of Flask-RESTX is quite straightforward and well-documented, each resource is represented by a class and can contain definitions of multiple HTTP methods. Endpoint description, input validation parser, response models etc. are defined in annotations of the resource class or HTTP method.

Example of Flask-RESTX annotations of the document embed endpoint from the finished microservice:

```
@api.route('/collection/<string:collection_name>/document')
@api.doc(params={'collection_name': 'Collection name'})
class CollectionDocuments(Resource):
    @api.doc(description="Embeds document from form data into
collection.")
    @api.response(model=res_models.embedResponse, code=201, description=
'OK')
    @api.response(model=res_models.error, code=400, description='Bad
model')
    @api.response(model=res_models.error, code=404, description='Resource
not found')
    @api.response(model=res_models.error, code=500, description='Server
error')
    @api.expect(req_parsers.embed_parser)
    def put(self, collection_name: str):
        # Endpoint logic
```

Input parameters documentation and validation are defined as a RequestParser object, as seen below for the same document embed endpoint as before:

```
embed_parser = reqparse.RequestParser(bundle_errors=True)
embed_parser.add_argument('file', location='files', type=FileStorage,
required=True)
embed_parser.add_argument('overlap_type', location='form', type=str,
choices=(common_models.OverlapType.SENTENCE.name,
common_models.OverlapType.PARAGRAPH.name), required=True)
embed_parser.add_argument('main_chunk_count', location='form',
type=inputs.positive, required=True)
embed_parser.add_argument('overlap_back_count', location='form',
type=inputs.natural, required=True)
embed_parser.add_argument('overlap_front_count', location='form',
type=inputs.natural, required=True)
embed_parser.add_argument('custom_metadata', location='form',
type=object, required=False)
```

The response model is documented by creating a response marshalling model, which provides an easy way to control what data is rendered in the response and is expected in the payload. As you can notice from the endpoint annotations from before, each response code can have a different response model defined. Document embed endpoint response model is defined as follows:

```
embedResponse = api.model(
    'EmbedResult',
    {
        'document_id': fields.String(description="Internal ID of the
embedded document.")
    }
)
```

Figure 5.5 shows the autogenerated response model documentation of the embed endpoint in Swagger UI, and figure 5.6 shows full interactive documentation of the endpoint, which can be therefore tested right in the browser.

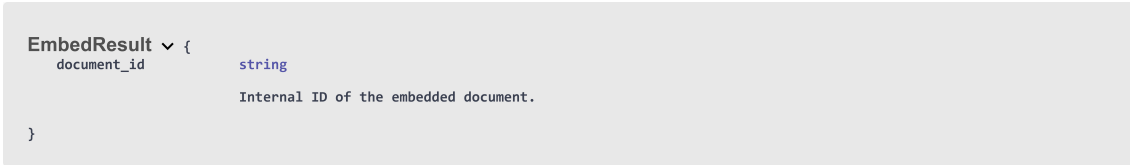


Figure 5.5. The documentation of document embed response model in Swagger UI.

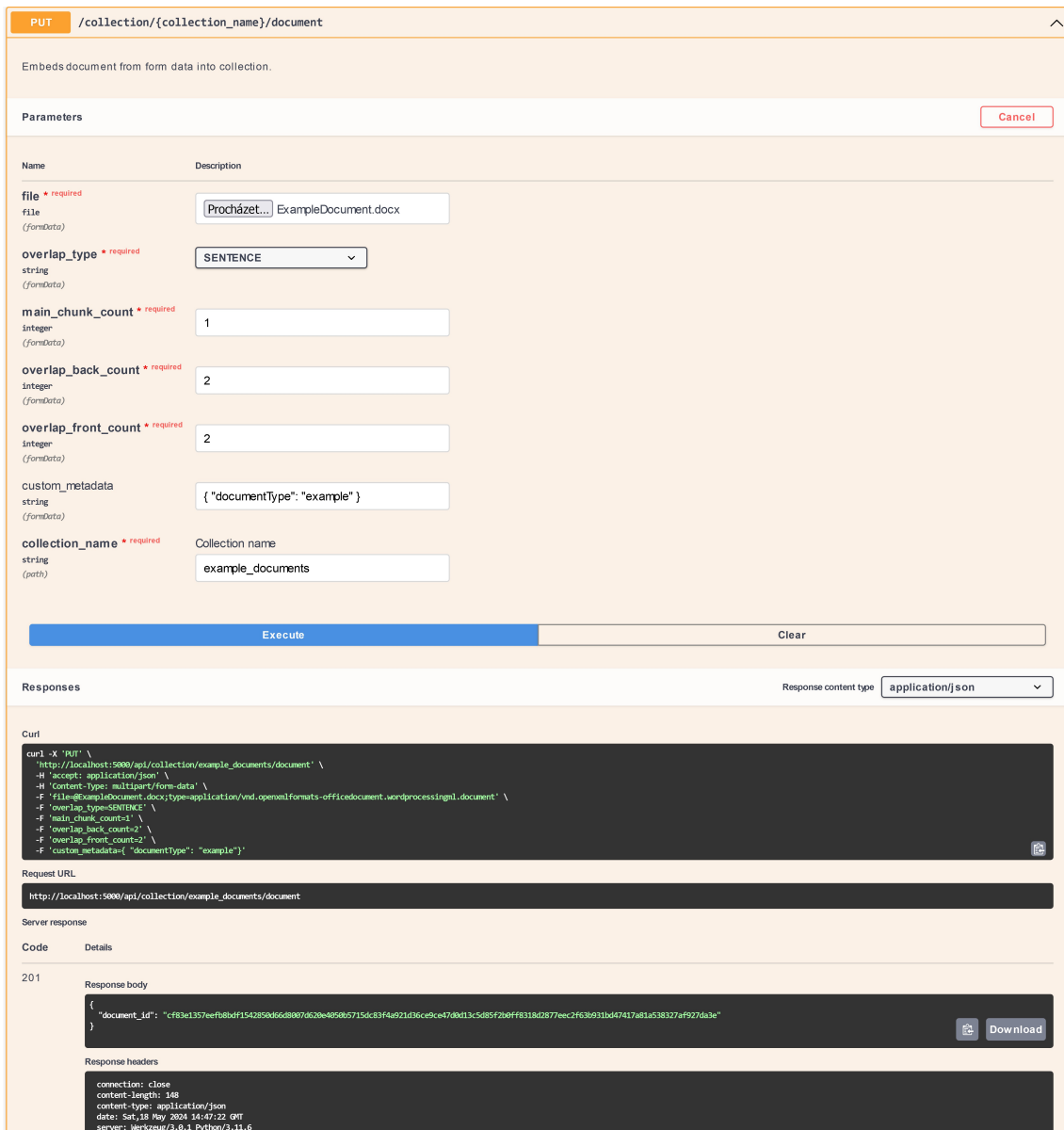


Figure 5.6. The embed endpoint documentation in Swagger UI.

The final autogenerated JSON file containing the OpenAPI specification is attached to the source code files and contains the specific request and response format and all HTTP response status codes (including error codes).

5.6 Implementation

In this section, I describe the implementation of parts of the microservice based on the requirements analysis and application design. I focus on the main steps in the implementation process and try not to go into unnecessary detail.

5.6.1 Development environment

For efficient development of the application, I used IDE (integrated development environment) Visual Studio Code [84]. It is a multi-platform open-source IDE developed by Microsoft, supporting JavaScript, TypeScript and Node.js at the base, while support for many other languages and runtime environments can be added using plugins (such as C++, C#, Java, Python, PHP, Go, or .NET) [85].

In addition to Visual Studio Code I occasionally used Notepad++ for simple text manipulation. Notepad++ is a free source code editor and Notepad replacement that supports several languages [86].

5.6.2 Project structure

Since the microservice project is small all source files are stored in a single directory. This allows for easy access to all required files and if the need for a more complicated structure ever arises it is quite straightforward to change it.

List of all project files:

```
vector_store_provider
|- Dockerfile
|- README.md
|- base_provider.py
|- chroma_connector.py
|- common_models.py
|- common_util.py
|- configuration_provider.py
|- docker-compose.yml
|- file_processor.py
|- gunicorn.conf.py
|- main.py
|- request_parsers.py
|- requirements.txt
|- response_models.py
```

Next, I describe all the project files grouped by their function.

5.6.2.1 Python microservice files

Common parts:

- `common_models.py` - Definitions of common data classes that are used across the whole microservice, most notably enum definitions and common wrapper data class which is used to convey additional processing information between layers in addition to the processed data itself. Referenced across the entire project.
- `common_util.py` - Definitions of common functions that can be used across the whole microservice, for example, a memory-efficient file hashing function. Referenced across the entire project.

■ API layer:

- `main.py` - Main microservice project file. Contains the setup of environmental variables and logging, Flask startup procedure, and Flask endpoint definitions. The main part of the API layer.
- `request_parsers.py` - Definition Flask_RESTX request parsers which are used to validate and document endpoint inputs. Referenced by `main.py`.
- `response_models.py` - Definition Flask_RESTX response models which are used to document endpoint outputs. Referenced by `main.py`.

■ Logical layer:

- `base_provider.py` - Definition of business logic methods, the main part of the logical layer. Called by the API layer, calls the data layer.
- `file_processor.py` - Definition of data extraction from files. Referenced by `base_provider.py`.

■ Data layer:

- `chroma_connector.py` - The main part of the data layer, contains the implementation of connection to the Chroma database and runs the embedding model. Called by the logical layer.

■ Other:

- `configuration_provider.py` - Logic for loading environmental variables. Called by `main.py`.
- `gunicorn.conf.py` - Configuration to load environmental variables correctly while the application is being managed by Gunicorn WSGI HTTP Server. Called internally by Gunicorn.
- `.env` - File storing environmental variables, loaded by `configuration_provider.py` and Gunicorn.
- `requirements.txt` - Python requirements definition, used to install the correct version of all dependencies (libraries) at once.

■ 5.6.2.2 Containerization files

- `Dockerfile` - Docker container image building specification for the microservice. Explained in detail in section 5.9.1.
- `docker-compose.yml` - Docker Compose specification to run the microservice Docker container with its dependencies. Explained in detail in section 5.9.1.

■ 5.6.2.3 Other files

- `README.md` - Contains basic information about the microservice and the steps for its deployment. Written in Markdown.

■ 5.6.3 Components

Logical splitting of the application into layers creates the main components of the application in addition to configuration, Chroma and embedding model components. Component diagram 5.7 shows the interconnection between the components and other working aspects of the microservice.

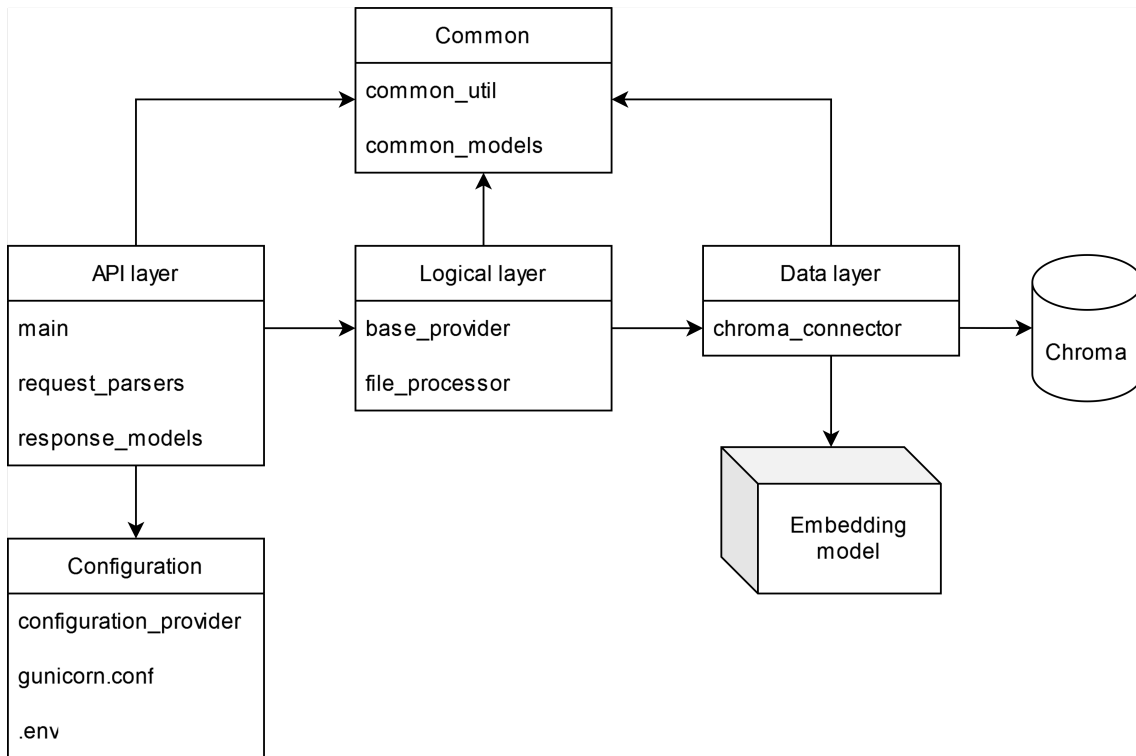


Figure 5.7. Microservice component diagram

5.7 Chosen embedding model

Many embedding models can be used, both small and large, runnable on the CPU or meant strictly for the GPU. It would be a really large task to compare them all, luckily a large number of them was ranked in MTEB: Massive Text Embedding Benchmark [87], and updated results are available on HuggingFace [26].

These embedding models range from millions to billions of parameters, from tens of MBs to tens of GBs of required runtime memory. Different models are also trained on different languages, and can only sometimes be used generally. This makes choosing the best model quite a hard task.

For the microservice, it would be ideal to have a single small embedding model that would decently embed both English and Czech text. This is because of lower available funds for running the model and the fact that the first language that needs to be supported is Czech, with English being the secondary. If the model is quite small it could also be run on the CPU at first, and only upgraded to a larger model running on a GPU if needed (which would mean re-embedding all the previously embedded documents).

Chroma uses all-MiniLM-L6-v2 model by default, which is a really small model with 22.7M (million) parameters, that needs about 90MB of memory, and is trained only on English data [45], and ranks at place 99 (out of 220) [26]. I have tried to find a similarly small model that is multilingual in the MTEB leaderboard, ideally a better-ranked one. Model multilingual-e5-small ranks at place 72 (out of 220) [26], has 118M parameters, needs about 440MB of memory, and supports 94 languages including Czech and English [88]. While this is about 5 times larger model it should still run decently on a CPU.

To determine whether the models could be run decently on the CPU for the initial deployment, I have decided to benchmark the embedding time on CPU vs GPU. Only the time for creating the embedding vector was measured, not the model initialization, warm-up time, clean-up time, or computing of results. For the embedded text I have used the first 186 tokens of Shakespeare’s Hamlet, available from the Massachusetts Institute of Technology webpage [89]. I created a Python script that first loads the model to be run on the CPU, does 10 warm-up embedding iterations, computes mean and deviation from 10000 iterations, cleans up the model, and then does the same for the GPU. This benchmarking script is a part of the source files.

First I tried benchmarking all-MiniLM-L6-v2 to gather data for comparison, and then multilingual-e5-small. Benchmark was run locally on Intel Core i5-7300HQ CPU, NVIDIA GeForce GTX 1050 GPU, and 24GB of DDR4 RAM. Results rounded to 2 decimal places are shown in table 5.5.

Model	Architecture	Mean runtime	Deviation
all-MiniLM-L6-v2	CPU	30.18ms	1.52ms
all-MiniLM-L6-v2	GPU	8.5ms	0.3ms
multilingual-e5-small	CPU	70.95ms	2.57ms
multilingual-e5-small	GPU	15.33ms	0.44ms

Table 5.5. Embedding model benchmark - CPU/GPU runtime.

From the results, we can see that running both models on CPU is about 4 times slower than on GPU, but since it still takes only tens of ms, with all-MiniLM-L6-v2 running about 2 times faster than multilingual-e5-small, which is 5 times larger. Based on these observations I have decided to use multilingual-e5-small, as it has better ranking all-MiniLM-L6-v2, supports multiple languages, and runs decently on CPU which decreases running costs.

5.8 Security

The microservice is designed to be run without being exposed to the internet publicly. While more advanced authentication and authorization can be used, the first version will be running on the same server as the main application (as specified in section 5.9.2), so the API itself can be run without any security features as long as its port is opened only locally and not to the internet.

In the future, if there is the need to deploy the microservice to a different server and be publicly accessible, there are many tools how to achieve this. For example, Flask-Security is a library, that allows quickly adding common security mechanisms to a Flask application, including session-based authentication, role and permission management, password hashing, basic HTTP authentication, token-based authentication, and more [90].

5.9 Deployment

While the microservice could be manually deployed as directly to a server, containerizing it and deploying it in a container is a better approach for a microservice. A container has everything needed to run the app and can be easily spun up or down for testing. This is how container deployment saves resources like storage, memory and processing

power and speeds up the CI/CD pipeline. Docker container deployment is a popular technology that gives developers the ability to construct application environments with speed at scale.[91]

■ 5.9.1 Containerization

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.[92]

■ 5.9.1.1 Dockerfile

To create a docker container, a Dockerfile containing the build instructions needs to be created. A Docker build consists of a series of ordered build instructions, each instruction in a Dockerfile roughly translates to an image layer [93].

Dockerfile:

```
FROM pytorch/pytorch:2.2.1-cuda12.1-cudnn8-runtime
RUN apt-get update && apt-get install ffmpeg libsm6 libxext6 -y
WORKDIR /app
COPY ./requirements.txt requirements.txt
RUN pip install --no-cache-dir --upgrade -r requirements.txt
COPY . .
CMD ["gunicorn", "--bind", "0.0.0.0:5000", "--timeout", "300",
    "main:app"]
```

The first instruction (FROM) in the Dockerfile specifies which base image should be used for the container, in this case, it is a pytorch image, which is based on Ubuntu 22.04 and contains all dependencies needed to run the embedding model on a GPU inside of the container.[94]

In the next layer, we install dependencies that are needed for the proper function of required Python libraries, specifically cv2 and its dependencies, which are needed for the unstructured library.

Then we create a working directory and install all required Python packages. Afterward, we can copy all application files into the working directory.

Finally, we create an entrance point for the container, which is running the Gunicorn WSGI server to serve the Flask application. The default parameters can be later overridden in a Docker Compose specification, as described in the following section.

■ 5.9.1.2 Docker compose

Having the microservice itself containerized isn't enough as we still need to deploy Chroma in headless mode somewhere and store its data. Since the microservice itself is containerized, it is logical to run Chroma in a container too. Multiple dependent Docker containers can be managed with Docker Compose.

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. It simplifies the control of the entire application stack, making it easy to manage services, networks, and volumes in a single, comprehensible YAML configuration file (docker-compose.yml). Then, with a single command, it is possible to create and start all the services from the configuration file.[95]

Docker Compose configuration file:

```

version: '3.9'
services:
  app:
    build: .
    command: "gunicorn --bind ${APP_HOST}:${APP_PORT} --timeout
${APP_TIMEOUT} main:app"
    ports:
      - "${APP_PORT}:${APP_PORT}"
    depends_on:
      chromadb:
        condition: service_healthy
    environment:
      - CHROMA_HOST=chromadb
  chromadb:
    image: chromadb/chroma:0.4.24
    volumes:
      - chromadb:/chroma/chroma
    environment:
      - IS_PERSISTENT=TRUE
      - ANONYMIZED_TELEMETRY=${ANONYMIZED_TELEMETRY:-TRUE}
      - CHROMA_HOST_PORT=${CHROMA_PORT}
    ports:
      - "${CHROMA_PORT}:${CHROMA_PORT}"
    healthcheck:
      test: [ "CMD", "/bin/bash", "-c", "cat < /dev/null >
/dev/tcp/localhost/${CHROMA_PORT}" ]
      interval: 10s
      timeout: 5s
      retries: 12
    volumes:
      chromadb:

```

The created configuration file specifies 2 containers: the microservice one (which is built according to the Dockerfile), and a Chroma container. Environmental variables for both these containers are automatically loaded from the .env configuration file. The Chroma container has a custom health check that needs to be met before the microservice starts, as the microservice is dependent on an already running Chroma instance. Chroma data is stored in a Docker persistent storage.

■ 5.9.2 Microservice deployment

The deployment of the finished containerized microservice can be done either manually (by building the container and running Docker Compose) or via continuous integration and continuous delivery/deployment (CI/CD) pipeline. Continuous integration refers to the practice of automatically and frequently integrating code changes into a shared source code repository. Continuous delivery and/or deployment is a 2 part process that refers to the integration, testing, and delivery of code changes.[96]

While the application source codes themselves are maintained in a GitHub repository for CI purposes, the CD pipeline is planned for the future and the first release of the microservice was done by manually building and deploying the container to a VPS (virtual private server).

The main application and other services are hosted on a VPS provided by Forpsi, which is a Czech hosting provider. Since the chosen embedding model will be running on a CPU (as explained in section 5.7) and the embedded data will need to be handled according to Czech regulations the microservice can be run on the same server. If the need arises to move the microservice to a different server it can be done quite easily because of the containerization.

The VPS to which the microservice is deployed has the following specifications [97]:

- Operating system: Debian 11
- CPU: 8 vCPU AMD
- RAM: 16 GB
- Storage: 160 GB NVMe SSD
- Traffic: 100 TB/month

As mentioned in the section 5.8, the microservice is not supposed to be exposed to the internet. The VPS is running Uncomplicated Firewall (ufw), which is a frontend for iptables and is particularly well-suited for host-based firewalls [98]. It is set up to allow connections only to the ports of the main application (80, 443) and SSH (22).

5.10 Finished microservice

In the end, the finished microservice is running on a VPS, only locally accessible to the main application and not exposed to the internet.

The microservice provides management of vector database collections and documents, embedding of new documents and querying the collections.

For embedding, 2 different chunking overlap methods are currently implemented. Both can have from 1 to n paragraphs as the main text of the chunk, with sentence overlap taking a specified number of last sentences from the previous chunk and/or first sentences from the next chunk (or more chunks, if one doesn't contain enough sentences), and paragraph overlap taking the specified number of paragraphs from before/after the main chunk text.

Both the microservice and Chroma are currently using logging into the default output, with plans to aggregate the logs onto a Syslog server from all the running containers and then use a log management and visualization tool (such as Graylog [99] or Grafana [100]) to allow easy access to the logs.

5.10.1 Integration with the main application

At the time of writing, the integration of the microservice to the workflow of the main application is not finished yet, full integration is planned in Q3 of 2024. The diagram 5.8 shows simplified services architecture, ie. the interconnection between implemented microservice, main application, and external connections of the project.

5.10.2 Future improvements

Since the microservice is not yet integrated with the main application there might be some changes required for easier integration.

In addition to that, there are already some features planned for the future, the most important one being offloading vector computation for document embedding to an external GPU, which would make it much faster, especially if more documents are needed to be embedded. The current plan is to create a minimalistic docker image with an embedding model and a singular API endpoint, which would compute and return

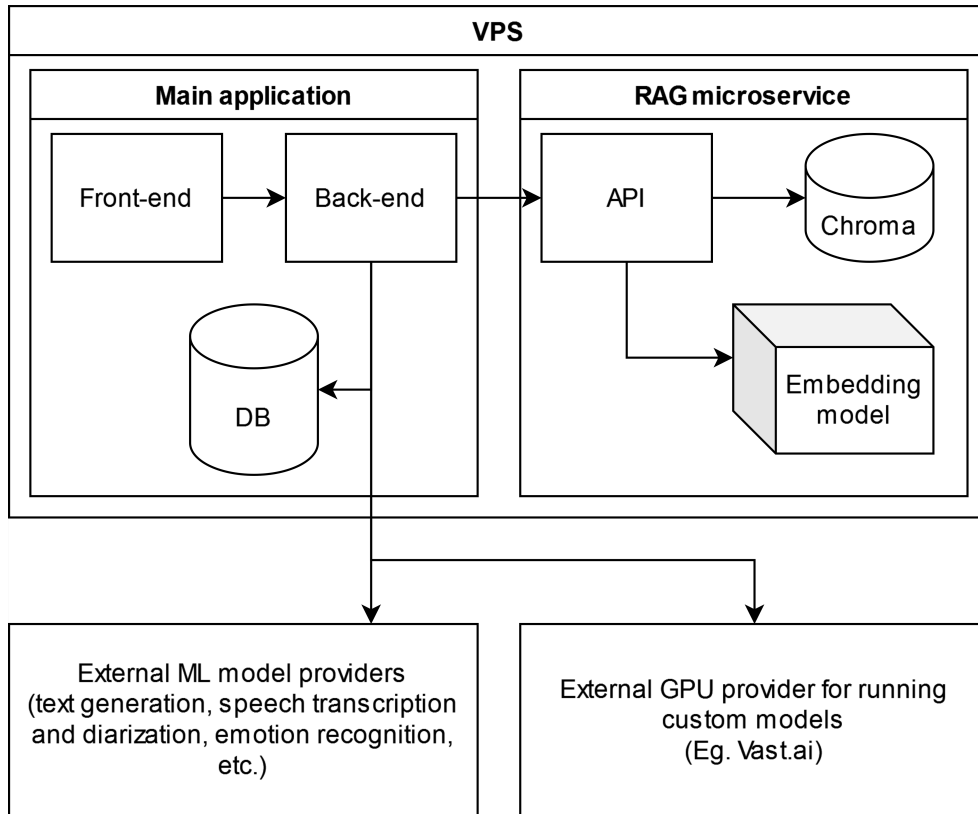


Figure 5.8. Services architecture diagram

vectors of send text. This container would be loaded on-demand on a GPU hosted by Vast.AI, which is a market-based cloud computing platform focused on reducing the costs and friction of compute-intensive workloads and enabling anyone to easily leverage large-scale GPU liquidity [101].

This would rapidly speed up the embedding of multiple large documents while keeping the GPU processing time cost down, as this model would be loaded only when needed.

Next, we would like to focus on improving our RAG pipeline results by evaluating its efficiency and making appropriate changes as discussed in section 4.6. This is both a time and resource-consuming task so it was not done as a part of this thesis. One of the easier changes could be the specification of the similarity threshold, if the threshold wouldn't be met by any context retrieved from the vector store, then the context wouldn't be used at all.

There is also the possibility of using different embedding models specifically trained for each language, which would greatly enhance the retrieval accuracy, at the cost of having to run multiple models at once.

Chapter 6

Comparison of LLM generation results with and without RAG

In this chapter, I show and compare LLM-generated outputs with and without RAG. Since the main application doesn't integrate the implemented RAG microservice yet, the process for obtaining the results was as follows:

1. Embed the document into the vector database using the finished microservice.
2. Use custom Python script to generate results using Claude 3 Sonnet. Context from the microservice is obtained by calling its API. The Jupyter Notebook with this testing implementation is part of the source files.
3. Perform qualitative analysis of the results.

Claude 3 Sonnet is a multimodal AI LLM model developed by Anthropic, balancing skills and speed, excelling in reasoning, multilingual tasks, and visual interpretation. It is the middle ground between their other Claude 3 models (Haiku and Opus), better and only slightly slower and pricier than Haiku, cheaper and faster but slightly worse than Opus.[102]

While Llama-2-13b was decent for testing the PoC solution, Sonnet is much more powerful even though it is still in beta and there are limits to the number of processed requests per minute. There is a discussion in our research team about whether to use it as the primary model for the main application.

For embedding, I have used 1 paragraph as a main chunk size, the front overlap consisting of the previous 3 sentences, and the back overlap consisting of the next 3 sentences. As a dataset, I have used an internal guideline of The National Qitline, which was provided for this project.

The guideline provided by The National Qitline is in the Czech language, so the conversation samples to query the vector database should be in Czech too because even though the embedding model is multilingual, measuring semantic similarity across multiple languages will yield worse results. This problem can be solved by translating either the embedded document into a general language like English and querying everything in English, or by translating the vector store query into the language of the document collection. Because the provided guideline is quite long, manual translation into English would take a lot of time, and machine translation is problematic because it is intended only for internal use within The National Qitline and shouldn't be distributed elsewhere.

In the end, I decided to use the following mix of conversations for comparison:

- 3 LLM generated sample conversations in English. Generated by GPT-4o, the LLM query will be in English, the vector store search query will be translated into Czech by Deepl and the result will be passed directly into the English LLM query because the large model shouldn't have a problem with multilingual processing.
- 5 LLM generated sample conversations in Czech. Generated by GPT-4o, the LLM query and vector store search query will be both in Czech.

- 2 pieces of real conversations in Czech, which were selected from the anonymized transcripts of the conversations dataset created by Patrik Jankuv, as explained in section 4.1. The quality of these anonymized transcripts varies (there are many misspellings and other issues), which represents a real conversation better than the LLM-generated samples. The LLM query and vector store search query will be both in Czech.

Since the guideline was provided to our team for internal use only I will not be showing the context retrieved from the vector store microservice.

While qualitative analysis can be subjective, using quantitative metrics (such as the RAG Triad of metrics, ROUGE, ARES, BLEU, or RAGAs) to measure RAG effectivity is difficult and requires manually labeled dataset, as described in section 4.6.

I show only one result of each category in this chapter, a complete list of all results is available in appendix A.

6.1 LLM prompts

In this section, I define the prompts used to query the LLM. The exact model version is “claude-3-sonnet-20240229“ with maximum generated tokens set to 512. I used message format for the prompt, meaning the LLM generation context is written as a dialogue between the user and the AI assistant, with the response being the next message to be written by the assistant.

For RAG I have used only the first and most similar result obtained from the vector store query.

In addition to the dialogue prompt, a detailed description of the task to achieve can be sent to the model as a system prompt. I use this to specify what to do and to include the RAG context retrieved from the microservice.

English system prompt without RAG:

```
You're an addiction specialist who answers to the patient in conversation. Answer the conversation in English in 2-3 sentences.
```

English system prompt with RAG context:

```
You're an addiction specialist who answers to the patient in conversation. Answer the conversation in English in 2-3 sentences. You may use the following context: {context}
```

Czech system prompt without RAG:

```
Jste adiktolog, který odpovídá pacientovi v rozhovoru. Odpovězte na konverzaci v češtině ve 2-3 větách.
```

Czech system prompt with RAG context:

```
Jste adiktolog, který odpovídá pacientovi v rozhovoru. Odpovězte na konverzaci v češtině ve 2-3 větách. Můžete použít následující kontext: {context}
```

6.2 Sample English conversation, GPT4o generated

Conversation text A.1.3:

Patient: I've been gambling a lot, and I'm losing more money than I can afford. It's putting a strain on my family.

Specialist: Gambling can be very consuming. How often do you find yourself gambling, and how much time and money are you spending?

Patient: Almost every day, and I've lost thousands in the past few months. I keep thinking I'll win it back, but it never happens.

Response generated without RAG:

I understand the urge to chase losses and try to win back the money you've lost through gambling. However, this tends to lead to an even deeper hole of debt and problems. Gambling addiction is a real issue that requires professional help to overcome. I would strongly encourage you to reach out to a gambling addiction hotline or counselor to get support in stopping this destructive cycle before it causes further financial and familial strain.

Response generated with RAG:

I understand the urge to try to win back losses, but continuing down that path often leads to digging the hole deeper. Gambling addiction is a real issue that can have severe financial and personal consequences. The fact that it is straining your family relationships is a major red flag. You've taken an important first step by being honest about the struggle. I would strongly encourage you to seek professional help from a gambling addiction counselor or program. They have strategies to help you regain control and repair the financial and relationship impacts. You don't have to fight this alone - there is support available.

Response comparison: The RAG result is more nuanced and specific than the LLM-only one.

6.3 Sample Czech conversation, GPT4o generated

Conversation text A.2.4:

Patient: Dobrý den, jsem závislý na marihuaně. Kouřím denně a mám pocit, že už to nezvládám bez ní.

Specialist: Dobrý den, děkuji, že jste přišel. Jak dlouho už takto kouříte?

Patient: Asi tři roky. Začalo to občas, ale teď už je to každý den.

Specialist: Jak se cítíte, když si marihuanu nedáte?

Patient: Jsem nervózní, nemůžu spát a mám problémy s jídlem.

Response generated without RAG:

Chápu, že pro vás odvykání marihuaně bude náročné, zejména kvůli fyzickým příznakům. Je však možné se závislosti zbavit s odbornou pomocí.

Doporučil bych vám kontaktovat nejbližší adiktologickou ambulanci nebo kliniku, kde vám poskytnou podporu během odvykacího procesu a naučí vás copingové strategie pro zvládání chutí. Se správným přístupem a podporou se dá závislost překonat. Nejste na to sám.

Response generated with RAG:

Rozumím, denní užívání marihuany může vést k závislosti a abstinčním příznakům. Je dobře, že jste to rozpoznal a přišel pro pomoc. Společně můžeme pracovat na léčbě závislosti a zvládnání odvykacích příznaků. Existují osvědčené metody jako kognitivně-behaviorální terapie, náhradní léčba a svépomocné skupiny, které vám mohou pomoci dosáhnout abstinence. Nejdůležitější je vaše odhodlání změnit svůj životní styl. Jsem tady, abych vás při tom podpořil a poskytl potřebné nástroje. Uděláme vše pro to, abyste se zbavil závislosti.

Response comparison: The RAG result is more nuanced and specific than the LLM-only one.

6.4 Sample Czech conversation, real

Conversation text A.3.1:

Patient: Dobrý den.
Specialist: Dobrý den. Národní linka pro odvykání.
Patient: My na [Date] máme domluvenou konzultaci. Máte, jakou chvíli čas.
Specialist: [Patient name]? Ano? Povídejte, co je nového teda o tý? Nebo jsme se neslyšeli.
Patient: No, nového od spolucující ohledně toho kouření nebo ostatní ještě.
Specialist: Ohledně toho kouření. Povídejte.
Patient: Ohledně kouření teda vlastně nic, no, to může prát stejně asi zhruba.
Specialist: To znamená teda těch 10 cigaret, jak jste kouřil jako 40, protože vlastně vy jste říkal minule, že, že jste vykouřil těch cca 10.
Patient: Ne, ne, krabičku, když jste měl nápla, tak jsem vykouřil krabičku, první den ho v kopce, [Date] [Date] jsem vykouřil zase půl kapku a půl a jsem si náplastí už radši nedával, protože to nemělo efekt žádnéj, takže tedkon cítil náplastí. Beru, jenom, když jedu na nějakou dlouhou cestu, ale třeba za psychiatrem nebo nakoupit autem nebo za psychologem prostě na delší cestu autem třeba, tak si dám, ale stejně. Kouřím, jo, tak si vás jdu, já poslal, stejně kouřím, takže.

Response generated without RAG:

Chápu, situace s kouřením se zatím příliš nezlepšila. Je dobře, že jste upřímný a popisujete mi realitu tak, jak opravdu je. Náhradní terapie s náplastmi vám zřejmě nevyhovovala. Pojďme se společně zamyslet, co by mohlo být efektivnější strategie pro snižování počtu vykouřených cigaret. Například postupné omezování, hledání náhradních činností nebo podpůrné metody jako je kognitivně-behaviorální terapie. Hlavně neztrácejme motivaci, odvykání je opravdu běh na dlouhou trať, kde jsou drobné neúspěchy naprosto normální. Pojďme hledat cesty, které vám budou nejvíce vyhovovat.

Response generated with RAG:

Chápu, situace se zatím moc nezměnila. I když náplastí krátkodobě pomáhaly, stále kouříte zhruba krabičku denně. Je dobré, že jste si uvědomil, že náplastí samy o sobě nejsou řešením a potřebujete kombinaci

více postupů. Neztrácejte motivaci, odvykání je proces, který chvíli trvá. Důležité je vytrvat a využít všechny možné cesty - náhradní terapii, behaviorální techniky i psychoterapeutickou podporu. Společně na tom můžeme dále pracovat a hledat cesty, které vám budou nejvíce vyhovovat. Hlavní je neztrácet naději a vydržet.

Response comparison: Both results are too similar to discern the better one.

6.5 Results discussion

In responses to both LLM-generated samples (6.2 and 6.3), we can see that using RAG increased the specificity of the response by better expression of the real approach that should be taken.

Sample 6.4 is inconclusive and nearly the same as without RAG.

Sample A.3.2 (which is based on an anonymized transcript of a real conversation) shows that the worse quality of the transcribed conversation combined with more data from RAG caused the LLM to hallucinate and the response is not relevant to alcohol addiction as it should be, but to prescription drug addiction instead.

In the end, 70% of the samples yielded better results with RAG when compared to LLM-only generation, except samples 6.4 (inconclusive), A.2.5 (inconclusive), and A.3.2 (hallucination).

Based on this observation we can conclude that the qualitative analysis shows improvement when using RAG when compared to direct LLM generation. Using quantitative metrics in the future will provide more insight into the specific improvements.

Chapter 7

Conclusion

This thesis has successfully demonstrated the theoretical knowledge, microservice design, and implementation required to leverage Retrieval-Augmented Generation technology for enhancing responses in addiction counseling scenarios. Through rigorous analysis and practical implementation, the study provided a comprehensive overview of RAG, from its foundational concepts to its application in a real-world setting, specifically focusing on conversations between addiction specialists and their clients.

Firstly, the thesis showcased an understanding of RAG technology. The literature review and theoretical framework covered the intricacies of natural language processing, large language models, and the benefits and challenges of integrating RAG into existing systems. This solid theoretical base was crucial in guiding the design and implementation phases of the project.

The design and implementation of the RAG-based microservice highlighted microservice architecture principles - the decision to utilize a microservice approach, as opposed to a monolithic architecture, ensuring flexibility, scalability, and ease of maintenance. The microservice was developed to manage vector database collections and documents, embed new documents, and query collections effectively. These capabilities are essential for the robust functioning of the RAG system within the broader application context.

The results discussion section provided an evaluation of the implemented RAG system. The analysis of LLM-generated responses, both with and without RAG, indicated significant improvements in specificity and relevance when RAG was employed. This qualitative assessment was supported by concrete examples, demonstrating the practical benefits of RAG in enhancing conversational AI applications in addiction counseling.

The thesis also acknowledged the limitations and potential future improvements for the RAG system. These include the offloading of vector computation to an external GPU to enhance processing speed, refining the RAG pipeline for better efficiency, and implementing more advanced data augmentation techniques. These future improvements are vital for further optimizing the performance and applicability of the implemented RAG system.

The successful completion of this project underscores the importance of integrating advanced AI technologies into specialized fields such as addiction counseling. By continually refining and improving these technologies, there is significant potential to offer more personalized, accurate, and supportive interactions, ultimately contributing to better outcomes for individuals seeking help.

In conclusion, this thesis has contributed to the field of AI and NLP by demonstrating the practical benefits of RAG technology. It provides a clear pathway for future research and development, highlighting the ongoing need for innovation and improvement in small AI-driven conversational systems. The implementation of the RAG microservice provided an example of an RAG application in a real-world setting.

References

- [1] LOCKE, Saskia, Anthony BASHALL, Sarah AL-ADELY, John MOORE, Anthony WILSON, and Gareth B. KITCHEN. Natural language processing in medicine: A review. *Trends in Anaesthesia and Critical Care*. 2021, Vol. 38, pp. 4-9. ISSN 2210-8440. Available from DOI <https://doi.org/10.1016/j.tacc.2021.02.007>.
<https://www.sciencedirect.com/science/article/pii/S2210844021000411>.
- [2] LEWIS, Patrick, Ethan PEREZ, Aleksandra PIKTUS, Fabio PETRONI, Vladimir KARPUKHIN, Naman GOYAL, Heinrich KÜTTLER, Mike LEWIS, Wen-tau YIH, Tim ROCKTÄSCHEL, Sebastian RIEDEL, and Douwe KIELA. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*.
- [3] GAO, Yunfan, Yun XIONG, Xinyu GAO, Kangxiang JIA, Jinliu PAN, Yuxi BI, Yi DAI, Jiawei SUN, Meng WANG, and Haofen WANG. *Retrieval-Augmented Generation for Large Language Models: A Survey*.
- [4] *Medium - Difference between AI, Machine Learning, NLP and Deep Learning* [online]. [cit. 2024-05-18]. Available from <https://becominghuman.ai/difference-between-ai-machine-learning-nlp-and-deep-learning-9f63066087f1>.
- [5] *DeepLearning.AI - Natural Language Processing* [online]. [cit. 2024-05-15]. Available from <https://www.deeplearning.ai/resources/natural-language-processing/>.
- [6] LIN, Steven Y., Megan R. MAHONEY, and Christine A. SINSKY. *Ten Ways Artificial Intelligence Will Transform Primary Care*. Available from DOI 10.1007/s11606-019-05035-1. Available from <http://dx.doi.org/10.1007/s11606-019-05035-1>.
- [7] OH, Kyo-Joong, Dongkun LEE, Byungsoo KO, and Ho-Jin CHOI. *A Chatbot for Psychiatric Counseling in Mental Healthcare Service Based on Emotional Dialogue Analysis and Sentence Generation*. Available from DOI 10.1109/mdm.2017.64. Available from <http://dx.doi.org/10.1109/MDM.2017.64>.
- [8] BELTRAMI, Daniela, Gloria GAGLIARDI, Rema ROSSINI FAVRETTI, Enrico GHIDONI, Fabio TAMBURINI, and Laura CALZA. *Speech Analysis by Natural Language Processing Techniques: A Possible Tool for Very Early Detection of Cognitive Decline?* Available from DOI 10.3389/fnagi.2018.00369. Available from <http://dx.doi.org/10.3389/fnagi.2018.00369>.
- [9] *Qliq - The Healthcare Chatbot* [online]. [cit. 2024-05-18]. Available from <https://www.qliqsoft.com/introducing-the-healthcare-chatbot-what-you-need-to-know-and-why>.
- [10] WANG, Jing, Huan DENG, Bangtao LIU, Anbin HU, Jun LIANG, Lingye FAN, Xu ZHENG, Tong WANG, and Jianbo LEI. *Systematic Evaluation of Research Progress on Natural Language Processing in Medicine Over the Past 20 Years: Bibliometric*

- Study on PubMed*. Available from DOI 10.2196/16816. Available from <http://dx.doi.org/10.2196/16816>.
- [11] COOK, Nigel, Anmol MULLINS, Raju GAUTAM, Sharath MEDI, Clementine PRINCE, Nishith TYAGI, and Jyothi KOMMINENI. *Evaluating Patient Experiences in Dry Eye Disease through Social Media Listening Research*. Adis Journals, 2019. Available from DOI 10.6084/M9.FIGSHARE.8108912. <https://link.springer.com/article/10.1007/s40123-019-0188-4>.
- [12] NI, Lin, Chenhao LU, Niu LIU, and Jiamou LIU. *MANDY: Towards a Smart Primary Care Chatbot Application*. Available from DOI 10.1007/978-981-10-6989-5_4. Available from http://dx.doi.org/10.1007/978-981-10-6989-5_4.
- [13] RUMSHISKY, A, M GHASSEMI, T NAUMANN, P SZOLOVITS, V M CASTRO, T H MCCOY, and R H PERLIS. *Predicting early psychiatric readmission with natural language processing of narrative discharge summaries*. Available from DOI 10.1038/tp.2015.182. Available from <http://dx.doi.org/10.1038/tp.2015.182>.
- [14] HE, Qiwei, Bernard P. VELDKAMP, Cees A. W. GLAS, and Theo de VRIES. *Automated Assessment of Patients' Self-Narratives for Posttraumatic Stress Disorder Screening Using Natural Language Processing and Text Mining*. Available from DOI 10.1177/1073191115602551. Available from <http://dx.doi.org/10.1177/1073191115602551>.
- [15] GIANFRANCESCO, Milena A., Suzanne TAMANG, Jinoos YAZDANY, and Gabriela SCHMAJUK. *Potential Biases in Machine Learning Algorithms Using Electronic Health Record Data*. Available from DOI 10.1001/jamainternmed.2018.3763. Available from <http://dx.doi.org/10.1001/jamainternmed.2018.3763>.
- [16] SALMAN, Shaeke, and Xiuwen LIU. *Overfitting Mechanism and Avoidance in Deep Neural Networks*.
- [17] SHEIKHALISHAHI, Seyedmostafa, Riccardo MIOTTO, Joel T DUDLEY, Alberto LAVELLI, Fabio RINALDI, and Venet OSMANI. *Natural Language Processing of Clinical Notes on Chronic Diseases: Systematic Review*. Available from DOI 10.2196/12239. Available from <http://dx.doi.org/10.2196/12239>.
- [18] *Wikipedia: The Free Encyclopedia - SNOMED CT* [online]. [cit. 2024-05-15]. Available from https://en.wikipedia.org/wiki/SNOMED_CT.
- [19] *IBM - What is retrieval-augmented generation?* [online]. [cit. 2024-01-20]. Available from <https://research.ibm.com/blog/retrieval-augmented-generation-RAG>.
- [20] *Gradient Flow - Techniques, Challenges, and Future of Augmented Language Models* [online]. [cit. 2024-05-18]. Available from <https://gradientflow.com/techniques-challenges-and-future-of-augmented-language-models/>.
- [21] *Medium - Retrieval-Augmented Generation (RAG) from basics to advanced* [online]. [cit. 2024-05-22]. Available from <https://medium.com/@tejpal.abhyuday/retrieval-augmented-generation-rag-from-basics-to-advanced-a2b068fd576c>.

-
- [22] *DataStax - Vector Embeddings: The Foundation of Data Representation* [online]. [cit. 2024-01-20]. Available from <https://www.datastax.com/guides/what-is-a-vector-embedding>.
- [23] DEVLIN, Jacob, Ming-Wei CHANG, Kenton LEE, and Kristina TOUTANOVA. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*.
- [24] *TensorFlow - word2vec* [online]. [cit. 2024-01-21]. Available from <https://www.tensorflow.org/text/tutorials/word2vec>.
- [25] *Allen Institute for AI - ELMo* [online]. [cit. 2024-01-21]. Available from <https://allenai.org/allennlp/software/elmo>.
- [26] *Hugging Face - Massive Text Embedding Benchmark (MTEB) Leaderboard* [online]. [cit. 2024-05-12]. Available from <https://huggingface.co/spaces/mteb/leaderboard>.
- [27] *deepset - The Beginner's Guide to Text Embeddings* [online]. [cit. 2024-05-18]. Available from <https://www.deepset.ai/blog/the-beginners-guide-to-text-embeddings>.
- [28] ROZADO, David. *Wide range screening of algorithmic bias in word embedding models using large sentiment lexicons reveals underreported bias types*. Available from DOI 10.1371/journal.pone.0231189. Available from <http://dx.doi.org/10.1371/journal.pone.0231189>.
- [29] *Pinecone* [online]. [cit. 2024-01-20]. Available from <https://www.pinecone.io/>.
- [30] *Chroma* [online]. [cit. 2024-01-20]. Available from <https://www.trychroma.com/>.
- [31] *LangChain - Introduction* [online]. [cit. 2024-01-20]. Available from https://python.langchain.com/docs/get_started/introduction.
- [32] JANKUV, Patrik. *Processing of dialogue data from the field of addiction counselling practice*. Available from <https://dSPACE.cvut.cz/handle/10467/113293>.
- [33] *National Library of Technology - Journal Search* [online]. [cit. 2024-01-20]. Available from <https://sfx.techlib.cz/sfx1cl41/journalsearch/ntk>.
- [34] *Addictology journal* [online]. [cit. 2024-01-20]. Available from <https://adiktologie-journal.eu>.
- [35] *National Institutes of Health - National Library of Medicine* [online]. [cit. 2024-01-20]. Available from <https://www.nlm.nih.gov/>.
- [36] *Firefox Addons - Link Gopher* [online]. [cit. 2024-01-20]. Available from <https://addons.mozilla.org/en-US/firefox/addon/link-gopher/>.
- [37] *JDownloader* [online]. [cit. 2024-01-20]. Available from <https://jdownloader.org/>.
- [38] *LangChain - PyPDFLoader* [online]. [cit. 2024-01-20]. Available from https://python.langchain.com/docs/modules/data_connection/document_loaders/pdf.
- [39] *Github - pdftotext* [online]. [cit. 2024-01-20]. Available from <https://github.com/jalan/pdftotext>.
- [40] *Github - PyMuPDF* [online]. [cit. 2024-01-20]. Available from <https://github.com/pymupdf/PyMuPDF>.

- [41] *Github - pdfminer* [online]. [cit. 2024-01-20]. Available from <https://github.com/euske/pdfminer>.
- [42] *Github - pdfplumber* [online]. [cit. 2024-01-20]. Available from <https://github.com/jsvine/pdfplumber>.
- [43] *Github - unstructured* [online]. [cit. 2024-01-20]. Available from <https://github.com/Unstructured-IO/unstructured>.
- [44] *LangChain - Chunking Best Practices for RAG Applications* [online]. [cit. 2024-01-20]. Available from <https://www.youtube.com/watch?v=uhVMFZjUOJI>.
- [45] *Hugging Face - all-MiniLM-L6-v2* [online]. [cit. 2024-01-20]. Available from <https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2>.
- [46] *Chroma - Integrations - Langchain* [online]. [cit. 2024-05-05]. Available from <https://docs.trychroma.com/integrations/langchain>.
- [47] *Medium - Evaluating RAG Applications with RAGAs* [online]. [cit. 2024-01-21]. Available from <https://towardsdatascience.com/evaluating-rag-applications-with-ragas-81d67b0ee31a>.
- [48] *Wikipedia: The Free Encyclopedia - Entity linking* [online]. [cit. 2024-01-21]. Available from https://en.wikipedia.org/wiki/Entity_linking.
- [49] ASAI, Akari, Zeqiu WU, Yizhong WANG, Avirup SIL, and Hannaneh HAJISHIRZI. *Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection*.
- [50] ART, Taylor. *J2EE and Beyond: Design Develop and Deploy World-Class Java Software*. Hoboken New Jersey U.S: Prentice Hall, 2002. ISBN 978-0-13-141745-8.
- [51] *Visual Paradigm - What is Use Case Diagram?* [online]. [cit. 2024-05-07]. Available from <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/>.
- [52] *Wikipedia - Software requirements specification* [online]. [cit. 2024-05-07]. Available from https://en.wikipedia.org/wiki/Software_requirements_specification.
- [53] *The App Solutions - Functional vs Non-functional Requirements: Main Differences & Examples* [online]. [cit. 2024-05-07]. Available from <https://theappsolutions.com/blog/development/functional-vs-non-functional-requirements/>.
- [54] *Atlassian - Microservices vs. monolithic architecture* [online]. [cit. 2024-05-05]. Available from <https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith>.
- [55] *Chroma - Usage Guide* [online]. [cit. 2024-05-05]. Available from <https://docs.trychroma.com/usage-guide>.
- [56] *LinkedIn - Pulse - SOAP vs REST: All you need to know* [online]. [cit. 2024-05-18]. Available from <https://www.linkedin.com/pulse/soap-vs-rest-all-you-need-to-know-luis-soares-m-sc->.

-
- [57] *IBM - What is a REST API?* [online]. [cit. 2024-05-08]. Available from <https://www.ibm.com/topics/rest-apis>.
- [58] *Raygun - SOAP vs REST vs JSON - a 2023 comparison* [online]. [cit. 2024-05-08]. Available from <https://raygun.com/blog/soap-vs-rest-vs-json/>.
- [59] *Chroma - Getting started* [online]. [cit. 2024-05-05]. Available from <https://docs.trychroma.com/getting-started>.
- [60] *Microsoft - Learn - ChromaClient* [online]. [cit. 2024-05-05]. Available from <https://learn.microsoft.com/en-us/dotnet/api/microsoft.semantickernel.connectors.chroma.chromaclient>.
- [61] *Python - What is Python? Executive Summary* [online]. [cit. 2024-05-05]. Available from <https://www.python.org/doc/essays/blurb/>.
- [62] *Flask* [online]. [cit. 2024-05-05]. Available from <https://flask.palletsprojects.com/en/3.0.x/>.
- [63] *FastAPI* [online]. [cit. 2024-05-05]. Available from <https://fastapi.tiangolo.com/>.
- [64] *Django* [online]. [cit. 2024-05-05]. Available from <https://www.djangoproject.com/>.
- [65] *Tornado* [online]. [cit. 2024-05-05]. Available from <https://www.tornadoweb.org/en/stable/>.
- [66] *Pyramid* [online]. [cit. 2024-05-05]. Available from <https://trypyramid.com/>.
- [67] *freeCodeCamp - Python Back-End Development – Handbook for Beginners* [online]. [cit. 2024-05-05]. Available from <https://www.freecodecamp.org/news/python-back-end-development-the-beginners-guide/>.
- [68] *Fortune - Education - AI programming languages power today's innovations like ChatGPT. These are some of the most popular* [online]. [cit. 2024-05-05]. Available from <https://fortune.com/education/articles/ai-programming-languages/>.
- [69] *MDN Web Docs - JavaScript language overview* [online]. [cit. 2024-05-08]. Available from https://developer.mozilla.org/en-US/docs/Web/JavaScript/Language_overview.
- [70] *Node.js - Introduction to Node.js* [online]. [cit. 2024-05-08]. Available from <https://nodejs.org/en/learn/getting-started/introduction-to-nodejs>.
- [71] *Moesif - Top 5 Node.js REST API Frameworks* [online]. [cit. 2024-05-08]. Available from <https://www.moesif.com/blog/api-product-management/api-analytics/Top-5-NodeJs-REST-API-Frameworks/>.
- [72] *Flask - Deploying to Production* [online]. [cit. 2024-05-08]. Available from <https://flask.palletsprojects.com/en/3.0.x/deploying/>.
- [73] *Werkzeug* [online]. [cit. 2024-05-08]. Available from <https://werkzeug.palletsprojects.com/en/3.0.x/>.
- [74] *Flask-RESTX* [online]. [cit. 2024-05-08]. Available from <https://flask-restx.readthedocs.io/en/latest/>.

- [75] *SentenceTransformers* [online]. [cit. 2024-05-08]. Available from <https://sbert.net/>.
- [76] *InstructorEmbedding* [online]. [cit. 2024-05-08]. Available from <https://github.com/xlang-ai/instructor-embedding>.
- [77] *Python - Logging* [online]. [cit. 2024-05-08]. Available from <https://docs.python.org/3/library/logging.html>.
- [78] *GitHub - python-dotenv* [online]. [cit. 2024-05-08]. Available from <https://github.com/theskumar/python-dotenv>.
- [79] *DEV Community - What is Application design?* [online]. [cit. 2024-05-08]. Available from <https://dev.to/hellonehha/what-is-application-design-53fa>.
- [80] *Medium - The Three Layered Architecture* [online]. [cit. 2024-05-08]. Available from <https://medium.com/@deanrubin/the-three-layered-architecture-fe30cb0e4a6>.
- [81] *Canarys - 3 Layered Architecture* [online]. [cit. 2024-05-18]. Available from <https://ecanarys.com/3-layered-architecture/>.
- [82] *GeeksforGeeks - REST API Architectural Constraints* [online]. [cit. 2024-05-08]. Available from <https://www.geeksforgeeks.org/rest-api-architectural-constraints/>.
- [83] *Swagger - About* [online]. [cit. 2024-05-08]. Available from <https://swagger.io/docs/specification/about/>.
- [84] *Visual Studio Code* [online]. [cit. 2024-05-10]. Available from <https://code.visualstudio.com/>.
- [85] *Visual Studio Code - Getting Started* [online]. [cit. 2024-05-10]. Available from <https://code.visualstudio.com/docs>.
- [86] *Notepad++ - What is Notepad++* [online]. [cit. 2024-05-10]. Available from <https://notepad-plus-plus.org/>.
- [87] MUENNIGHOFF, Niklas, Nouamane TAZI, Loïc MAGNE, and Nils REIMERS. MTEB: Massive Text Embedding Benchmark. *arXiv preprint arXiv:2210.07316*. arXiv, 2022. Available from DOI 10.48550/ARXIV.2210.07316. <https://arxiv.org/abs/2210.07316>.
- [88] *Hugging Face - Multilingual-E5-small* [online]. [cit. 2024-05-12]. Available from <https://huggingface.co/intfloat/multilingual-e5-small>.
- [89] *Massachusetts Institute of Technology - The Tragedy of Hamlet*. [cit. 2024-05-12]. Available from <https://shakespeare.mit.edu/hamlet/full.html>.
- [90] *Flask-Security* [online]. [cit. 2024-05-12]. Available from <https://flask-security-too.readthedocs.io/en/stable/>.
- [91] *Avi Networks - Container Deployment* [online]. [cit. 2024-05-10]. Available from <https://avinetworks.com/glossary/container-deployment/>.
- [92] *Docker - Use containers to Build Share and Run your applications* [online]. [cit. 2024-05-10]. Available from <https://www.docker.com/resources/what-container/>.
- [93] *Docker - Layers* [online]. [cit. 2024-05-10]. Available from <https://docs.docker.com/build/guide/layers/>.
- [94] *DockerHub - Pytorch image - Layers - 2.2.1* [online]. [cit. 2024-05-10]. Available from

-
- <https://hub.docker.com/layers/pytorch/pytorch/2.2.1-cuda12.1-cudnn8-runtime/images/sha256-11691e035a3651d25a87116b4f6adc113a27a29d8f5a6a583f8569e0ee5ff897>.
- [95] *Docker - Docker Compose overview* [online]. [cit. 2024-05-10]. Available from <https://docs.docker.com/compose/>.
- [96] *Red Hat - What is CI/CD?* [online]. [cit. 2024-05-10]. Available from <https://www.redhat.com/en/topics/devops/what-is-ci-cd>.
- [97] *FORPSI - Classic VPS* [online]. [cit. 2024-05-10]. Available from <https://www.forpsi.com/virtual/>.
- [98] *Debian - Wiki - Uncomplicated Firewall* [online]. [cit. 2024-05-10]. Available from <https://wiki.debian.org/Uncomplicated%20Firewall%20%28ufw%29>.
- [99] *Graylog* [online]. [cit. 2024-05-18]. Available from <https://graylog.org/>.
- [100] *Grafana* [online]. [cit. 2024-05-18]. Available from <https://grafana.com/>.
- [101] *Vast.ai - Global GPU Market* [online]. [cit. 2024-05-18]. Available from <https://vast.ai/>.
- [102] *Anthropic - Introducing the next generation of Claude* [online]. [cit. 2024-05-19]. Available from <https://www.anthropic.com/news/claude-3-family>.

Appendix A

RAG results

A.1 English conversations generated by GPT4o

A.1.1 Conversation 1

Conversation text:

Patient: I think I might have a problem with alcohol. I drink almost every night, and it's starting to affect my work and relationships.
Specialist: Thank you for sharing that. Can you tell me more about your drinking habits? For example, how much do you typically drink in a night?
Patient: Usually a bottle of wine, sometimes more. It helps me relax, but lately, it feels like I need more to get the same effect.

Response generated without RAG:

It sounds like you are developing a tolerance to alcohol, which is a common sign of alcoholism or alcohol use disorder. Drinking every night and needing to increase the amount to feel the desired effect are red flags that your alcohol use has become problematic. I would recommend speaking to an addiction counselor or your doctor for an assessment and to get support in cutting back or quitting drinking. Early intervention can help prevent further negative impacts on your work, relationships and health.

Response generated with RAG:

I'm sorry to hear that alcohol has started impacting important areas of your life like work and relationships. Developing a tolerance and needing to drink more to achieve the desired effects are signs of alcohol dependence. However, there is help available. As an addiction specialist, I would recommend speaking to your doctor or a counselor about getting support to cut back or quit drinking. With the right treatment plan and support system, many people are able to regain control over alcohol use. You've taken an important first step in acknowledging there may be an issue. Recovery is possible, one day at a time.

Response comparison: The RAG result is more nuanced and specific than the LLM-only one.

A.1.2 Conversation 2

Conversation text:

Patient: I've been taking painkillers ever since my surgery last year, and I can't seem to stop. I'm worried I'm addicted.
Specialist: It's good that you're reaching out. Painkillers can be very

addictive, especially after long-term use. How often are you taking them now?

Patient: Almost daily. I tried to stop, but the withdrawal symptoms were too intense.

Response generated without RAG:

I understand this is a difficult situation. Daily use of painkillers and experiencing withdrawal symptoms when trying to quit are signs of physical dependence and addiction. It's important to seek professional help to taper off painkillers safely and receive support for managing withdrawal. With medical supervision and a structured treatment plan, overcoming painkiller addiction is possible. Please don't try to quit cold turkey on your own as that can be dangerous.

Response generated with RAG:

I understand this is a difficult situation. Developing a dependence on painkillers after surgery is quite common. It's wise to seek professional help to safely manage withdrawal symptoms and treat the addiction. I would recommend speaking to an addiction specialist or your doctor about a medically supervised detox program. They can provide support and medications to ease the withdrawal process. Overcoming painkiller addiction is challenging, but very possible with the right treatment plan and support system in place. The most important first step is reaching out for help, which you've already taken.

Response comparison: The RAG result is more nuanced and specific than the LLM-only one.

■ A.1.3 Conversation 3

Conversation text:

Patient: I've been gambling a lot, and I'm losing more money than I can afford. It's putting a strain on my family.

Specialist: Gambling can be very consuming. How often do you find yourself gambling, and how much time and money are you spending?

Patient: Almost every day, and I've lost thousands in the past few months. I keep thinking I'll win it back, but it never happens.

Response generated without RAG:

I understand the urge to chase losses and try to win back the money you've lost through gambling. However, this tends to lead to an even deeper hole of debt and problems. Gambling addiction is a real issue that requires professional help to overcome. I would strongly encourage you to reach out to a gambling addiction hotline or counselor to get support in stopping this destructive cycle before it causes further financial and familial strain.

Response generated with RAG:

I understand the urge to try to win back losses, but continuing down that path often leads to digging the hole deeper. Gambling addiction is a real issue that can have severe financial and personal consequences. The fact that it is straining your family relationships is a major red flag. You've taken an important first step by being honest about the struggle.

I would strongly encourage you to seek professional help from a gambling addiction counselor or program. They have strategies to help you regain control and repair the financial and relationship impacts. You don't have to fight this alone - there is support available.

Response comparison: The RAG result is more nuanced and specific than the LLM-only one.

A.2 Czech conversations generated by GPT4o

A.2.1 Conversation 1

Conversation text:

Patient: Dobrý den, pane doktore, myslím, že mám problém s alkoholem. Piju každý den a už to nezvládám kontrolovat.
 Specialist: Dobrý den, rozumím. Jak dlouho už pijete takto pravidelně?
 Patient: Asi tak poslední dva roky. Ze začátku to bylo jen občas, ale teď už je to denně.
 Specialist: Chápu. Jaký máte pocit, když se pokusíte nepít?
 Patient: Jsem nervózní, podrážděný a mám problém se soustředit. Je to, jako by mi něco chybělo.

Response generated without RAG:

To, co popisujete, zní jako tělesná závislost na alkoholu. Je dobře, že jste si tohoto problému vědom a přicházíte pro pomoc. Léčba závislosti na alkoholu zahrnuje řadu kroků, jako je detoxikace, terapie a svépomocné skupiny. Bude to vyžadovat trpělivost a odhodlání, ale s správnou podporou je vážně možné závislost překonat a vrátit se k životu bez alkoholu. Nechci na vás naléhat, ale doporučil bych vám zvážit léčbu.

Response generated with RAG:

Je zřejmé, že u vás už došlo k závislosti na alkoholu. Nedostatek alkoholu v těle vám působí odvykací příznaky jako je nervozita, podrážděnost a nesoustředěnost. Přestat pít najednou po delší závislosti může být zdravotně rizikové, proto bych doporučil podstoupit odbornou ústavní léčbu. Tam vám zkušení odborníci pomohou bezpečně detoxikovat organismus a následně vám nabídnou vhodný způsob dlouhodobé léčby závislosti, ať už formou doléčovacích pobytů, terapií nebo svépomocných skupin. S jejich pomocí a vaší motivací je zvládnutí závislosti reálné. Není třeba se stydět a nepodcenit váš problém.

Response comparison: The RAG result is more nuanced and specific than the LLM-only one.

A.2.2 Conversation 2

Conversation text:

Patient: Dobrý den, mám pocit, že jsem závislý na hazardních hrách. Skoro každý den hraju a dělám dluhy.
 Specialist: Dobrý den, to je vážný problém. Jak dlouho už máte pocit, že vás hazardní hry ovládají?

Patient: Asi poslední rok. Nejdřív to bylo jen pro zábavu, ale teď už to nemůžu přestat dělat.

Specialist: A jaké to má dopady na váš každodenní život a vztahy?

Patient: Kvůli dluhům se hádám s rodinou a začínám mít problémy v práci.

Response generated without RAG:

Je důležité, abyste si závislost na hazardních hrách přiznal a vyhledal odbornou pomoc. Bez ní bude velmi obtížné se z tohoto bludného kruhu dostat sám. Můžeme vám doporučit kontakty na organizace, které se na léčbu patologického hráčství specializují. Najít správnou motivaci a oporu v rodině vám může hodně pomoci překonat tuto závislost. Nebojte se o podporu požádat, rozhodně na to nejste sám.

Response generated with RAG:

Chápu, že je to pro vás velmi obtížná situace. Závislost na hazardních hrách může mít vážné následky, ale s odbornou pomocí se dá překonat. Důležité je si přiznat problém a rozhodnout se něco s tím udělat. Mohl bych vám doporučit kontaktovat některou z organizací, které se specializují na léčbu návykového chování. Tam vám mohou poskytnout potřebnou podporu a terapii, abyste se dokázal ze závislosti dostat. Nejste v tom sám a není pozdě věci změnit k lepšímu. Hlavně nevzdávejte úsilí a věřte, že to zvládnete.

Response comparison: The RAG result is more nuanced and specific than the LLM-only one.

■ A.2.3 Conversation 3

Conversation text:

Patient: Dobrý den, mám problém se závislostí na lécích na předpis.

Užívám je denně, i když už bych neměl.

Specialist: Dobrý den, chápu. Které léky užíváte a jak dlouho?

Patient: Jsou to hlavně léky na bolest. Užívám je asi dva roky. Začalo to po operaci, ale teď už je potřebuji pořád.

Specialist: A co se stane, když se pokusíte léky nevízt?

Patient: Mám silné bolesti a nemůžu se soustředit na nic jiného.

Taky jsem velmi podrážděný.

Response generated without RAG:

Rozumím, že jde o závažnou závislost, která vážně ovlivňuje váš život. Byl bych velmi opatrný s náhlým vysazením léků, mohlo by to být pro vás nebezpečné. Doporučil bych vám vyhledat odbornou pomoc v nějaké léčebně závislosti nebo alespoň poradit se s lékařem. Detoxifikace pod lékařským dohledem a následná léčba vám mohou velmi pomoci získat závislost pod kontrolu a předejít zdravotním komplikacím. Nemusíte to zvládnout sám, existují účinné léčebné postupy. Nebojte se požádat o pomoc.

Response generated with RAG:

Rozumím, že jde o složitou situaci. Vzhledem k délce užívání těchto léků a příznakům při jejich vynechání se pravděpodobně vyvinula závislost. Pokračování v tomto režimu však může vést k dalšímu prohlubování problémů. Pomocť vám může odborná léčba závislosti, kde vám odborníci pomohou bezpečně léky vysadit a zvládnout abstinční příznaky.

Současně vám nabídnou podporu při hledání jiných cest ke zmírnění bolesti. Není nutné se stavu vzdávat, existují cesty jak se závislosti pracovat pod odborným vedením. Můžeme vám doporučit vhodné adiktologické služby ve vašem okolí.

Response comparison: The RAG result is more nuanced and specific than the LLM-only one.

■ A.2.4 Conversation 4

Conversation text:

Patient: Dobrý den, jsem závislý na marihuaně. Kouřím denně a mám pocit, že už to nezvládám bez ní.

Specialist: Dobrý den, děkuji, že jste přišel. Jak dlouho už takto kouříte?

Patient: Asi tři roky. Začalo to občas, ale teď už je to každý den.

Specialist: Jak se cítíte, když si marihuanu nedáte?

Patient: Jsem nervózní, nemůžu spát a mám problémy s jídlem.

Response generated without RAG:

Chápu, že pro vás odvykání marihuaně bude náročné, zejména kvůli fyzickým příznakům. Je však možné se závislosti zbavit s odbornou pomocí. Doporučil bych vám kontaktovat nejbližší adiktologickou ambulanci nebo kliniku, kde vám poskytnou podporu během odvykacího procesu a naučí vás copingové strategie pro zvládání chutí. Se správným přístupem a podporou se dá závislost překonat. Nejste na to sám.

Response generated with RAG:

Rozumím, denní užívání marihuany může vést k závislosti a abstinenčním příznakům. Je dobře, že jste to rozpoznal a přišel pro pomoc. Společně můžeme pracovat na léčbě závislosti a zvládání odvykacích příznaků. Existují osvědčené metody jako kognitivně-behaviorální terapie, náhradní léčba a svépomocné skupiny, které vám mohou pomoci dosáhnout abstinence. Nejdůležitější je vaše rozhodnutí změnit svůj životní styl. Jsem tady, abych vás při tom podpořil a poskytl potřebné nástroje. Uděláme vše pro to, abyste se zbavil závislosti.

Response comparison: The RAG result is more nuanced and specific than the LLM-only one.

■ A.2.5 Conversation 5

Conversation text:

Patient: Dobrý den, myslím, že jsem závislý na sociálních médiích. Trávím tam celé dny a zanedbávám ostatní věci.

Specialist: Dobrý den, to může být vážný problém. Jak dlouho už trávíte tolik času na sociálních médiích?

Patient: Asi poslední dva roky. Cítím se tam dobře, ale pak si uvědomím, že jsem ztratil celý den.

Specialist: A jaký vliv to má na vaše každodenní aktivity a vztahy?

Patient: Zanedbávám práci, školu a dokonce i přátele. Jsem často podrážděný, když nemám přístup k telefonu.

Response generated without RAG:

Chápu, že sociální média pro vás představují určitý únik, ale jejich nadměrné používání může mít vážné negativní důsledky. Je dobré, že jste si toho vědom. Abychom se s tím mohli vypořádat, bylo by vhodné stanovit si nějaká pravidla ohledně času stráveného on-line, najít si nějaké další zájmy a aktivity offline a případně vyhledat odbornou pomoc, například formou poradenství nebo terapie. Je důležité včas začít se závislostí pracovat, aby nenarušovala váš život.

Response generated with RAG:

Je dobré, že si uvědomujete negativní dopady nadměrného užívání sociálních médií. Závislost na nich může skutečně narušovat každodenní fungování a vztahy. Můžeme se společně zaměřit na strategie, jak čas strávený online více omezit. Pomocí může například časové omezení, blokování aplikací nebo nahrazení nutkání jinou aktivitou. Hlavně to nevzdávejte, s trochou úsilí se dá závislost překonat. Jsem tu pro vás, abych vám poskytl podporu.

Response comparison: Both results are too similar to discern the better one.

A.3 Czech conversations from the anonymized transcript dataset

A.3.1 Conversation 1

Conversation text:

Patient: Dobrý den.
 Specialist: Dobrý den. Národní linka pro odvykání.
 Patient: My na [Date] máme domluvenou konzultaci. Máte, jakou chvilku čas.
 Specialist: [Patient name]? Ano? Povídejte, co je nového teda o tý? Nebo jsme se neslyšeli.
 Patient: No, nového od spolucující ohledně toho kouření nebo ostatní ještě.
 Specialist: Ohledně toho kouření. Povídejte.
 Patient: Ohledně kouření teda vlastně nic, no, to může prát stejně asi zhruba.
 Specialist: To znamená teda těch 10 cigaret, jak jste kouřil jako 40, protože vlastně vy jste říkal minule, že, že jste vykouřil těch cca 10.
 Patient: Ne, ne, krabičku, když jste měl nápla, tak jsem vykouřil krabičku, první den ho v kopce, [Date] [Date] jsem vykouřil zase půl kapku a půl a jsem si náplastí už radši nedával, protože to nemělo efekt žádný, takže teďkon cítil náplastí. Beru, jenom, když jedu na nějakou dlouhou cestu, ale třeba za psychiatrem nebo nakoupit autem nebo za psychologem prostě na delší cestu autem třeba, tak si dám, ale stejně. Kouřím, jo, tak si vás jdu, já poslal, stejně kouřím, takže.

Response generated without RAG:

Chápu, situace s kouřením se zatím příliš nezlepšila. Je dobře, že jste upřímný a popisujete mi realitu tak, jak opravdu je. Náhradní terapie s náplastmi vám zřejmě nevyhovovala. Pojdme se společně zamyslet, co by mohlo být efektivnější strategie pro snižování počtu vykouřených cigaret. Například postupné omezování, hledání náhradních činností nebo podpůrné

metody jako je kognitivně-behaviorální terapie. Hlavně neztrácejme motivaci, odvykání je opravdu běh na dlouhou trať, kde jsou drobné neúspěchy naprosto normální. Pojďme hledat cesty, které vám budou nejvíce vyhovovat.

Response generated with RAG:

Chápu, situace se zatím moc nezměnila. I když náplasti krátkodobě pomáhaly, stále kouříte zhruba krabičku denně. Je dobré, že jste si uvědomil, že náplasti samy o sobě nejsou řešením a potřebujete kombinaci více postupů. Neztrácejte motivaci, odvykání je proces, který chvíli trvá. Důležité je vytrvat a využít všechny možné cesty - náhradní terapii, behaviorální techniky i psychoterapeutickou podporu. Společně na tom můžeme dále pracovat a hledat cesty, které vám budou nejvíce vyhovovat. Hlavní je neztrácet naději a vydržet.

Response comparison: Both results are too similar to discern the better one.

■ A.3.2 Conversation 2

Conversation text:

Patient: Dobrý den.
 Specialist: Národní linka pro odvykání. Prosím.
 Patient: Dvě vozíky já jsem měl vůbec. My jsme vám dneska volali, jo, před.
 Adiktolog: [Patient name].
 Patient: Jasně, my jsme vám volali, protože jsme měli domluvený s vámi hovor na. Předpokládám, že na [Date].
 Adiktolog: Neměli.
 Patient: Neměli, zněli, tak já teda mám namřené, že, že máme domluvený hovor na [Date]. V [Date], ale pokud se vám to nehodí, tak se můžeme domluvit na jiné. Ne? Pohodí se to, že si promluvit.
 Adiktolog: Dobře, dobře, tak jak se vám daří?
 Patient: No, to je tak někde na opase třeba [Date] pij, už se lidé, ne? Nebo zase čtyři piju, tři dny ne? A furt to jde. Doch, ale nikomu neshodím, nikomu nevadím. Takhle se, když vám vypijte, tak se se mnou mu tak nebaví.

Response generated without RAG:

Rozumím, že pití alkoholu pro vás představuje problém, se kterým si nevíte rady. Je dobře, že jste se rozhodl vyhledat odbornou pomoc. Alkoholismus je nemoc, kterou lze léčit, ale vyžaduje to velkou vůli a podporu okolí. Navrhuji, abychom se nejprve společně zaměřili na pochopení vašich důvodů k pití a rizikových situací, které k němu vedou. Postupnými kroky pak můžeme pracovat na změně životního stylu a zvládání chutí na alkohol. Hlavně neztrácejte naději, s odhodláním a trpělivostí se dá problém zvládnout.

Response generated with RAG:

Rozumím, že užívání léků bez lékařského předpisu a ve větším množství pro vás představuje problém, se kterým si nevíte rady. Je dobré, že jste se rozhodl situaci řešit. Přesto bych vám doporučil obrátit se nejprve na svého praktického lékaře nebo přímo na odborníky v některém z adiktologických poraden nebo stacionářů. Tam vám pomohou nastavit vhodný odvykací režim a zajistí potřebnou péči a podporu během celého procesu.

Na naší lince bohužel nemáme dostatečné kapacity na vedení dlouhodobé odvykací terapie. Mohu vám ale poskytnout kontakty na nejbližší zařízení, kde vám odborně poradí.

Response comparison: LLM-only is correct, while the RAG one is incorrect, because of model hallucination and subsequently irrelevant response.

Appendix B

List of abbreviations

RAG	Retrieval-augmented generation
NLP	Natural language processing
PoC	Proof-of-concept
LM	Language model
LLM	Large language model
AI	Artificial intelligence
NLU	Natural language understanding
NLG	Natural language generation
CNN	Convolutional neural network
RNN	Recurrent neural network
EHR	Electronic health record
API	Application programming interface
DB	Database
ID	Unique identifier
REST	Representational state transfer
SOAP	Representational state transfer
HTTP	Hypertext transfer protocol
JS	JavaScript
WSGI	Web server gateway interface
CRUD	Create, read, update and delete
UI	User interface
IDE	Integrated development environment
CI	Continuous integration
CD	Continuous delivery/deployment
VPS	Virtual private server
ufw	Uncomplicated firewall



Appendix C

List of attachments

The attachments of this thesis are divided into the following folders:

- `model_benchmark` - Scripts used for benchmarking performance of embedding models when running on CPU in comparison to running on GPU
- `poc_solution` - Scripts used as a part of the PoC solution
- `results_comparison` - Scripts used for the generation of final results
- `swagger_specification` - Final autogenerated Swagger OpenAPI specification of the microservice
- `vector_store_provider` - Source codes of the microservice

Appendix D

List of used software

- Python - Programming language
- Visual Studio Code - IDE (integrated development environment)
- Notepad++ - Text editor
- Enterprise Architect - Diagrams
- Draw.io - Diagrams
- Docker - Deployment
- Postman - Endpoint testing
- Deepl - Translation, AI tool
- Grammarly - Text correction, AI tool
- QuillBot - Plagiarism check, AI tool
- GPT3, GPT4o - Text restructuring, generation of sample conversations, AI tool
- Upscayl - Image upscaling, AI tool
- Llama-2-13b-hf - AI model used for PoC implementation
- Claude 3 Sonnet - AI model used for generating responses for results comparison