



# Chatbot Project

---

Satakunta University of Applied Sciences – RoboAI Academy

**Assignment:** Chatbot to answer questions about cyber security directive

**Project group:** Yilin Wang & Shristi Bhandari

**Supervisor:** Toni Aaltonen

## Starting point/problem

Our project is a part of Kytee. Kytee project aims to ensure that industry complies with the requirements of the revised EU Information Security Directive (NIS2). The company needs to ensure that they are suitable for the NIS2 standard. Kytee have a website that have some questions to help them confirm whether they have completed the standard, as well as some steps. Our project creates a chatbot that can answer questions about NIS2.

## Project definition

This project focused on creating a chatbot designed to assist users by answering their questions accurately, based on the content of the NIS2 document. The chatbot ensures that all responses are relevant and aligned with the standards outlined in NIS2. Additionally, an API was developed to enable related parties to integrate and utilize the chatbot seamlessly. As a result of this project, the client receives a reliable and accessible tool to support NIS2 compliance efforts.

## Project team responsibilities and learning objectives

This project is planned and executed by Yilin Wang and Shristi Bhandari, and together we have agreed to share responsibilities as follows:

Yilin Wang:

- Project Plan
- Build RAG with Claude & OpenAI assistant model
- Build API
- Github related work
- Test models

Shristi Bhandari:

- Research about RAG
- Build the first version of RAG model
- Test and evaluate the models
- Communicate about project related parties
- Test API
- Some other translation
- Project Report

This project provided an opportunity to learn about Large Language Models (LLMs) and Natural Language Processing (NLP). Yilin Wang improved technical skills in developing Retrieval-Augmented Generation (RAG) systems using Claude and OpenAI assistant models, gaining hands-on experience in API development, model testing, and managing GitHub repositories.

Shristi Bhandari focused on deepening knowledge of RAG systems through research and implementation, enhancing skills in evaluating and testing models, improving communication with stakeholders, and contributing to project documentation and translation tasks. Together, the project allowed us to develop collaboration, documentation, teamwork skills, and effectively communicate project requirements.

## Techniques

This project has a wide range of techniques and new things that the project team will get to know and learn before we can move on to the next practical step. The following is a brief description of these techniques from a project perspective:

### 1. RAG

In this project our data input is documents, chatbot should find related content in documents and generation the answer. If we train models by ourselves, there will be a problem. After every time we add a new document, we need to train the model again, it will need much time and resources.

### 2. Ollama

It was the first version of the RAG model that we used in the project. The reason behind using this technology is: it was easy to run and test the questions using this model and it does not require any API access when we use it. It is also low at cost. It runs locally, so if have a private document, Ollama is the best choice.

### 3. OpenAI Assistant [gpt-4o & gpt-4o-mini]

OpenAI Assistant has a method called "search in files", it is based on RAG, and it is easy to use, we can only use OpenAI Assistant, it has already packaged many methods, we just need to use it directly. In this project we used gpt-4o & gpt-4o-mini 2models, gpt-4o have more accurate, gpt-4o-mini fast and low-cost.

### 4. Claude [Haiku, Sonnet, opus]

Claude is the latest technology and is much faster than the rest of the technologies used by us before. It generates human-like responses to users' prompts and questions. It has a different model which is much faster and better at responding.

### 5. LumiOpen

LumiOpen have be trained many Finnish data Input, it means it have better understand for Finnish language compare with other models.

### 6. FastAPI

FastAPI is a modern web framework for building APIs with Python, Python is our develop language in this project. FastAPI is eased to use, it can significantly reduce our development time, as our project only has two members, low code helps us save time.

## Github link for this project

<https://github.com/RoboAI-Kytee/roboai-cyberlab-cybersecurity-self-assessment-tool-chatbot/>

## Actors of the project

- Aaltonen Toni: Supervisor
- Jönkkäri Teemu: Support Document, Evaluate the models
- Tähti Leena: Evaluate the models
- Viljanen Jani: Evaluate the models
- Aalto Juha: helped evaluate the translation work

## Progress of the project

## Phase 1: Exploration and Initial Research

The first two weeks the project concentrated in research of the RAG process, this proceeded in the following way:

Document → Chunk → Embedding → VectorDB → LLM Answer.

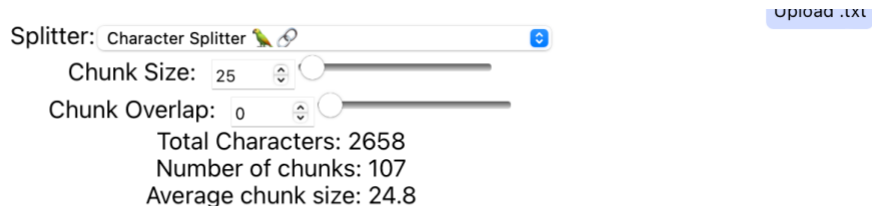
- **Document:** Source text is provided as input.
- **Chunk:** The document is split into smaller, manageable chunks of text.
- **Embedding:** Each chunk is converted into a high-dimensional vector using embedding techniques, capturing semantic meaning.
- **VectorDB:** The embeddings are stored in a vector database to enable efficient similarity searches.
- **LLM Answer:** When a query is made, relevant chunks are retrieved from the VectorDB and used as context for the Language Model (LLM) to generate an answer.

Explored Ollama + LangChain for local PDF chatting, and exploration of embedding model. We used OpenAI's text-embedding-ada-002 model to embed the NIS2 directive documents. This model was chosen because it effectively converts text into high-dimensional vectors, capturing the semantic meaning of the content. These vectors are essential for performing similarity searches and generating accurate responses in the chatbot.

The **text-embedding-ada-002** model is designed to transform text into numerical representations, which allows the system to understand the context of the documents and retrieve the most relevant information when a user asks a question. We selected this model for its high accuracy and efficiency in embedding text, as well as its ability to handle large-scale data.

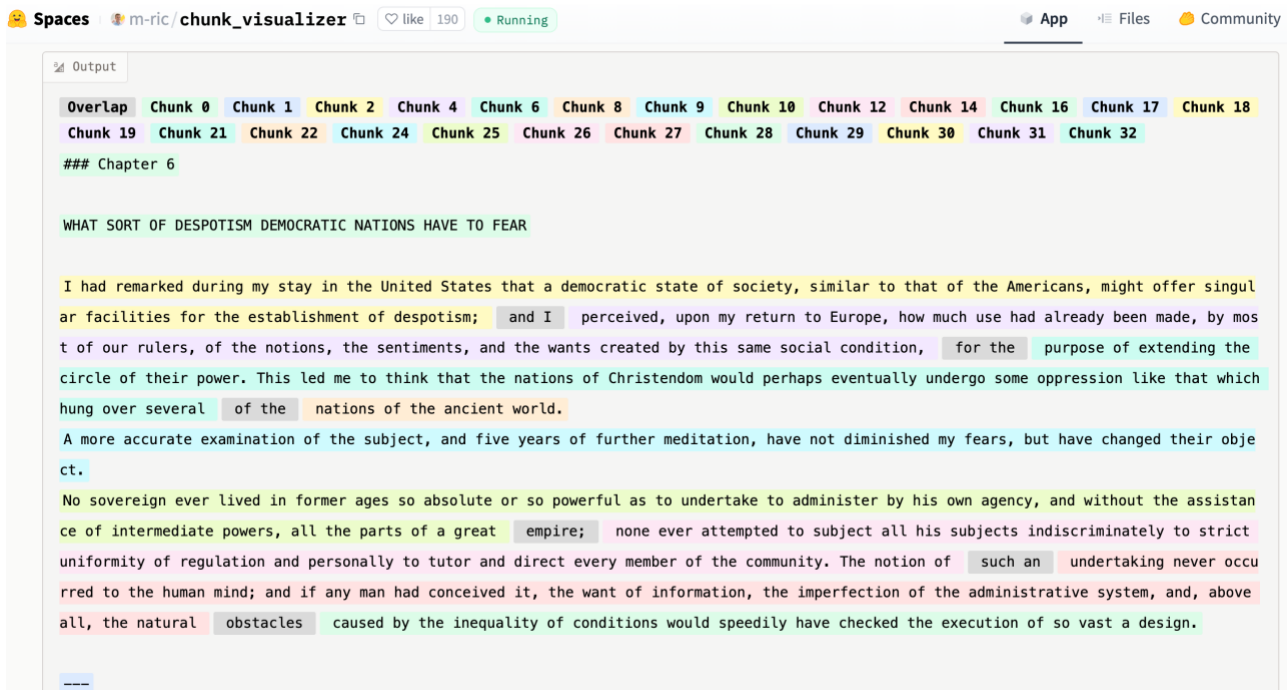
We valuated chunk size and text Splitting Performance. Investigated chunk size and text splitter performance using tools like Chunkviz and Chunk\_visualizer. Both "[Chunkviz](#)" and "[Chunk\\_visualizer](#)" are tools designed to help users understand and visualize how text is divided into chunks, which is particularly useful in natural language processing (NLP) and machine learning applications.

Chunkviz:



One of the most important things I didn't understand about the world when I was a child is the degree to which the returns for performance are superlinear.

Chunk\_visualizer:



Purpose for chunking is splitting documents into smaller chunks ensures that the most relevant portion of the text can be found during similarity searches. If documents are not split, the entire document embedding might ignore important details, making retrieval less precise.

Experimented with different chunk sizes and overlaps for testing question accuracy. Finally, we decided to use chunk size 2000 and chunk overlap 200, which was the most suitable splitting sizes for our document.

### Challenges:

During the project, several challenges were encountered. One issue is inconsistent answers when using different chunk sizes. Additionally, there were problems with Ollama, as it often failed to produce accurate responses. The model can answer the question, but the answer varies when the same question is asked multiple times. We tried adjusting the chunk size and setting the temperature to 0 (to ensure consistent answers), but this approach did not work either.

### Phase 2: Experimentation with Models

In the third week, we focused on tested different models, tested translate tool and tested prompt. This is explained in more detail in the following paragraphs.

OpenAI is a common model and easier to use than Ollama. OpenAI can handle text generation and language translation, and we tested its translate is good. So in our project we can only focus on RAG and openai can help us do the both generation and translate.

Switched Ollama to OpenAI API for better performance of the model. We used English documents as our data input, and also selected 10 different questions in the PDF to test Olama's query results. However, its performance was not good. We tried different chunks, but it may have omissions in different questions. For example, a question has 5 key points, and depending on the chunk, it may only answer three or four. Compared to OpenAI, its answer quality is not high. We believe that OpenAI is more accurate than it, so we gave up using it as LLM.

Add Context Query Prompts to ensure answers were focused on the documents. Also to make sure that the generated answers are relevant to the provided documents, context query prompts

were added. This helps the model understand and focus on specific sections or information within the documents, improving answer accuracy.

To translate Finnish and English, we also tested 10 different tools for that using Finnish questions and Finnish documents, and we used tools like google translate microsoft translator, pons translator, Linguee translator, Mymemory translator, Yandex translator, QcriTranslator translator, Papago translator, Libre translator, LumiOpen and Chatgpt.

Top 3 Translation Tools:

Translation Tools	Rank
ChatGPT	1
Libre Translator and Linguee translator	2
Google Translate	3

LumiOpen is an LLM, and it be trained use many Finnish input, it may mean, it have better understand Finnish than other LLM. But there are some disadvantages of Lumiopen. First, the dataset trained on Lumiopen is focused on Finnish and Norwegian, which makes it unable to complete the translation task. However, our input file is in English, so it cannot understand the content of the document well. And we tried to use more GPU's [4 GPUs -136GB] but the model was very slow (more than 30mins).

#### Challenges:

- The models didn't work well with Finnish data.
- Processing was slow with LumiOpen Model, and we faced issues using multiple GPUs effectively.

#### Phase 3: Fine-Tuning and Workflow Optimization

In the fourth week, we focused on use Claude model in our project, fine-tuning the model and try to use Redis caching.

##### Use Claude Model

Claude is the latest technology which is the most accurate and efficient tool we used than Ollama and LumiOpen. We tried three different models of Claude which are opus, Haiku and Sonnet, and all those versions were almost faster than the rest of the models we tried in the past. Claude generally gives clear and useful answers in a short period of time.

##### Optimizing Response Speed with Redis Caching

Redis caching stores previously computed results in memory, which significantly reduces the time it takes to retrieve answers for high frequency asked queries. This is crucial for systems that need to provide quick responses, especially when dealing with large datasets or frequent user queries.

When we use Redis, we evaluate the similarity of problem, identify issues in the database that are similar to user problems, and then retrieve the corresponding answers. However, we encountered some issues:

- Due to insufficient user issue data, it is difficult for us to assess which issues are high-frequency.
- Even if similar answers are found, we still need a large language model to generate answers because the questions are not exactly the same.
- It is difficult for us to evaluate whether the answers in the database accurately match the user's questions.

##### Tested Claude System Prompts

Better Context: Provides more relevant and accurate responses by setting clear context.

Consistency: Ensures reliable and predictable behavior across interactions.

Customization: Allows for personalized responses based on user needs.

More information about Claude System Prompts: <https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/system-prompts#how-to-use-system-prompts>

#### Improved Code Efficiency

In the initial code, there was an issue where the vector database was generated every time the code was executed, resulting in long execution times. We modified the code to first check if the database already existed. If it did, we used it directly; if not, we created a new database. This way, the database is generated only when the code is first run, making it more efficient.

#### Challenges:

- Claude's performance varied with prompts, requiring multiple testing.
- Speed issues persisted despite GPU optimizations.

#### Phase 4: Evaluation and Refinement

The fifth weeks we focused on use OpenAI Assistant in our project, after that we tested models and comparison different model. Then we developed new system prompts and made the processing pipelines simpler.

##### Use OpenAI Assistant Model

The OpenAI Assistant model is highly capable of understanding and generating human-like text. It is easy to use, we can only use OpenAI Assistant, it is already packaged with many methods, we just need to use it directly. It is also very accurate in answering within short periods of time. We used two models which were gpt4o and gpt4o-mini in our case. Gpt4o is the most powerful with a higher accuracy rate whereas got4o-mini is the fastest version.

##### Model Testing and Comparison

In this phase, different models were tested and compared to evaluate their performance with the Retrieval-Augmented Generation (RAG) approach. The models tested included Claude, OpenAI, and LumiOpen, along with Claude's PDF support. The goal was to assess how well each model handled document-based queries and provided accurate answers.

Developed new system prompts and made the processing pipelines simpler.

We tested different levels of prompts in our case. We experienced that with different prompts the model answers differently. If we have good and well-structured prompt instruction, the model answers in the same way and vice versa.

Planned to build API support for Teemu and integrate the model into AIServer, allowing easier integration and communication with the system.

#### Challenges:

Improving accuracy needed more precise prompt design.

#### Phase 5: Getting Ready for Deployment

In the sixth week, we evaluated the model and determined the final version code to be released. Next these are explained with more details.

All previous code was organized into a structured folder, and a README file was added to provide clear guidance and easy reference for users. Additionally, we tested OpenAI Assistant models (GPT-4o and GPT-4o-mini) and Claude models (Haiku and Sonnet) using relevant documents to evaluate their performance. The comparison involved assessing the performance of Claude (Haiku and Sonnet) and OpenAI Assistant models (4o and 4o-mini) on specific questions using three distinct prompts—two provided by Teemu and one created by us. These tests evaluated both the



speed and quality of the results, providing valuable insights into the models' efficiency and effectiveness.

The following are the test results. OpenAI Assistant 4o Model with first edition prompt is the best one. 4o, 4o mini and Claude Sonnet was really close each other. Haiku was fastest but also weakest of them all.

- [9,50] 4o / first edition Prompt
- [9,14] Claude 3.5 Sonnet / first edition Prompt
- [9,07] 4o mini / first edition Prompt
- [8,93] 4o mini / second edition Prompt
- [8,93] Claude3.5 Sonnet / second edition Prompt

Additionally, to enhance user convenience, we developed and provided APIs that allow users to utilize the model without needing to understand its specific code implementation, making the system more user-friendly and accessible.

### **Phase 6: Documentation and finishing work**

The final phase of the project focused on wrapping up all remaining tasks to ensure the work was complete, well-documented, and ready for use. First, we assessed the resources required for the project, including GPU and CPU needs, to ensure the system's scalability and efficiency. Next, the project report was finalized to document the progress, challenges, and outcomes comprehensively. The completed code was uploaded to GitHub, accompanied by a detailed document explaining the code's structure, functionality, and step-by-step instructions on how to use it. These steps ensured that the project was thoroughly documented and accessible for future use or further development.

## **Solution**

We Created a chatbot for this project. This chatbot should answer questions from users, and its answer should related document about NIS2, and the answers should be correct. We gave related parties an API to help them easy to use it.

## **Summary**

This project aimed to create a chatbot to help companies comply with the EU's updated cybersecurity directive (NIS2). The chatbot provides accurate answers to questions about NIS2 by using a technology called Retrieval-Augmented Generation (RAG), which retrieves relevant information from documents to generate responses.

We tested models like OpenAI, Claude, and LumiOpen (for Finnish) and faced challenges with accuracy, Finnish translations, and speed. After six weeks of research, testing, and improving, we delivered a working chatbot that's ready to help companies comply with NIS2.