Implementacija jezičnog asistenta temeljenog na umjetnoj inteligenciji za brodarsku industriju

Špoljar, Ivan

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS No. 419

IMPLEMENTATION OF AN ARTIFICIAL INTELLIGENCE-BASED CHATBOT FOR THE SHIPPING INDUSTRY

Ivan Špoljar

Zagreb, June 2024

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS No. 419

IMPLEMENTATION OF AN ARTIFICIAL INTELLIGENCE-BASED CHATBOT FOR THE SHIPPING INDUSTRY

Ivan Špoljar

Zagreb, June 2024

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Zagreb, 04 March 2024

MASTER THESIS ASSIGNMENT No. 419

Student:	**Ivan Špoljar (0036523646)**
Study:	Computing
Profile:	Software Engineering and Information Systems
Mentor:	assoc. prof. Marina Bagić Babac

Title: **Implementation of an artificial intelligence-based chatbot for the shipping industry**

Description:

The goal of this thesis is the implementation of a chatbot based on artificial intelligence and adapted to the specific needs of a shipping company, with a focus on answering questions relevant to its activity. The main task is the application of large language models in the process of creating a functional chatbot. By adapting the previously trained model and limiting access to only publicly available data of Croatian shipping companies, the paper will demonstrate the practical application of theoretical concepts in the real world. The chatbot's performance will be evaluated using appropriate metrics in order to assess its effectiveness and reliability in providing relevant answers to the questions asked.

Submission date: 28 June 2024

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 419

Pristupnik:	**Ivan Špoljar (0036523646)**
Studij:	Računarstvo
Profil:	Programsko inženjerstvo i informacijski sustavi
Mentorica:	izv. prof. dr. sc. Marina Bagić Babac

Zadatak: **Implementacija jezičnog asistenta temeljenog na umjetnoj inteligenciji za brodarsku industriju**

Opis zadatka:

Cilj ovog diplomskog rada je implementacija jezičnog asistenta temeljenog na umjetnoj inteligenciji prilagođenog specifičnim potrebama brodarske tvrtke, s fokusom na odgovaranje na pitanja relevantna za njezinu djelatnost. Glavni zadatak je primjena velikih jezičnih modela u procesu izrade funkcionalnog asistenta. Kroz prilagodbu prethodno naučenog modela te ograničavanje pristupa samo javno dostupnim podacima hrvatskih brodarskih tvrtki, rad će demonstrirati praktičnu primjenu teorijskih koncepta u stvarnom svijetu. Odgovarajućim metrikama provest će se evaluacija performansi jezičnog asistenta kako bi se procijenila njegova učinkovitost i pouzdanost u pružanju relevantnih odgovora na postavljena pitanja.

Rok za predaju rada: 28. lipnja 2024.

Contents

Figures

Tables

Introduction

In this time which we live in, technology is more widespread than ever before, and it just keeps going forward. The goal of this advancement is to make it easier for people to use technology and to make technology that is human-like. It got to the point where robots can imitate humans and computers can make conversations with them. One of the popular and frequently used types of technology that has capability of making conversations with people is the chatbot. As the interest of companies in incorporating a chatbot in their business grows, in this paper, the process of developing a chatbot is demonstrated.

This paper presents and explains the basic terms and concepts related to the creation of a chatbot, more precisely Vectors, Embedding, Grounding, LLM and RAG. Additionally, two text embedding models and two text generation models are combined, and the performances of their combinations are compared.

The chatbot that is developed is restricted to answering only specific questions about the shipping company and its activity and answers only in Croatian language.

This chatbot is implemented as a web application with two pages, home page and page for inserting new knowledge. The home page contains a messaging interface for communication with chatbot and a container for sources in which there are displayed texts from which chatbot composed his answer. Page for inserting new knowledge contains text area for inserting text, input field for web page URL, and file input field.

1. Related works

In this chapter, other papers related to implementation of a chatbot with a specific purpose will be presented.

1.1. AI Based Chatbot for Educational Institutions

This paper is written by Oswalt Manoj S, Jason Jose P, Johans Olivia A, and Katyayani T R to promote smooth interaction between users and educational institutions, and to spread necessary information. Their chatbot can perform question answering about academic programs, faculty details, and institutional policies. The chatbot was developed by integrating web scraping, tokenization, vectorization, and the large language model GPT-2. Their work proved that there is a potential in using an AI to revolutionize educational environments and support systems. (S et al., 2024)

1.2. Implementation of an NLP-Driven Chatbot and ML Algorithms for Career Counseling

In this paper, written by Anuja Deshpande, Aryan Dubey, Arya Dhavale, Ankita Navatre, Uma Gurav, and Amit Kumar Chanchal, the chatbot for educational and career selection is proposed. Their proposed system evaluates students' strengths, weaknesses, interests, and aptitudes to advise the student about the best choice that would suit him based on interests, capabilities, and opportunities. They tried using different machine learning algorithms among which the decision tree performed the best and showed great potential for further development. (Deshpande et al., 2024)

1.3. Iterative design and implementation of a chatbot for sexual and reproductive health counseling in Peru

This paper is written by Norma Leon Lescano, Eiriku Yamao, Elizabeth Xiomara Valladares Sánchez, and Miguel Angel Pablo Estrella Santillan, with the purpose of bringing reliable information about sexual and reproductive health to young people. Their chatbot is designed to follow guidelines and tools from the Ministry of health in Peru and the World Health Organization to counsel young people about sexual and reproductive health. They conducted a survey about acceptance of the use of the chatbot service for that subject and the acceptance rate was 65%, which they want to increase. (Lescano et al., 2022)

2. Methodology

In this chapter, the basic terms and concepts needed to understand and develop a functional chatbot are listed and explained. Also, the architecture of the application is described, and the choice of used models is explained.

2.1. Chatbot

Chatbot is a computer program that simulates and processes human conversation and makes it possible for humans to interact with digital devices as if they were communicating with a real person. (Oracle, 2020) AI chatbot is a chatbot which uses some sort of AI technology like NLP, ML or LLM.

2.2. Vectors

A vector is a mathematical structure with a size and a direction which represents a point in space. In programming, a vector is an array containing numerical values. (Schwaber-Cohen, 2023) Vectors are contained in a vector space which enables performing certain operations on them, for instance, vector addition and scalar multiplication. Vector space also provides a framework for studying vectors which gives us the ability to manipulate their quantities, not just adding and subtracting, but also comparing. It is precisely the comparison that enables us to perform similarity searches on vectors. Similarity search is in fact search for the distance, smaller the distance, higher the similarity. (Descartes, 2024)

2.3. Embedding

Embeddings are multi-dimensional representations, in numeric format, of words, phrases, sentences, images, graphs, or any form that has meaning to humans. (Berger, 2023) Embeddings are basically functions that map other types of data to vectors. (Descartes, 2024)

2.4. Grounding

Grounding is a process in which the responses from Large Language Models are limited to the knowledge we want them to have. Large Language Models possess a large amount of knowledge and have an understanding of sentence construction, it is so because Large Language Models are trained on large sets of data, most of which is irrelevant when we want them to have a specific use. Grounding Large Language Models causes greater accuracy in giving answers and their better quality as it ensures that the answers are relevant to a specific use for which we use them. The most common techniques for grounding are Retrieval-Augmented Generation, which will be explained and used in this paper, and Fine-tuning. (Berger, 2023)

2.5. Large Language Models (LLM)

A Large Language Model is a type of Artificial Intelligence which is able to understand natural language and put meaningful sentences together. Language Models work in the way they predict a probability for each possible next word which makes them in fact statistical models. The most important part of using Large Language Models is giving them good instructions, in a natural language, on how to generate an answer, and that is called the prompt. The prompt is responsible for the behavior of Large Language Models because it instructs them on what task they should do, which knowledge they should use, and in what type and what format they should return the answer. Large Language Models can perform various tasks, from text generation, text classification, translating languages, generating embeddings, and some more. (Keng, 2023)

Today's Large Language Models use the transformer architecture, unlike earlier Natural Language Processing Models which used to use convolutional neural network and recurrent neural network. The transformer architecture is different because it utilizes self-attention which enables parallel processing of different segments of the input sequence and that causes the model to understand dependencies between segments on different positions in a sequence. (Luo et al., 2023)

Self-attention is a mechanism which consists of two parts, scaled dot-product attention and multi-head attention. Scaled dot-product attention is a function with the use of which the calculated output contains values to which the corresponding weights have been assigned, based input which consists of query vectors (Q), key vectors (K), and value vectors (V). The process of calculating attention is as follows. First, the dot product of query vectors and key vectors is calculated. Dot product is calculated as shown in equation $x \times y = \sum_i x_i \times y_i$. Next, the result of dot product is scaled to ensure that attention weights don't get too low or too high. Scaling is performed dividing earlier calculated dot product with squared root of the dimension of key vectors ($\sqrt{d_k}$). After the scaled dot product is calculated, the softmax function is performed on it to convert the dot product into probabilities which sum equals one. (Vaswani et al., 2017)

The softmax function is performed in three steps. First, each value in the result matrix is used to raise the number e, i.e. Euler's number, to the power of that value. For each value the function e^{value} is performed and from that the new matrix is obtained. Next, the sum of each column from matrix from the previous step is calculated. The final step is to divide each value from the matrix with the sum of the column to which the value belongs to. (Dey, 2024) The dot product of that new matrix and value vectors is then calculated, and the result is the scaled dot-product attention. The function for scaled dot-product attention can be written as $Attention(Q, K, V) = softmax\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) \times V$.

Multi-head attention is a set of scaled dot-product attentions being calculated in parallel in each available head, with Q, K, and V divided into h parts. The function for multi-head attention is $Multi - head(Q, K, V) = Concat\ (head_1, \dots, head_h) \times W^0$, where W^0 is learned in the process, and concat signifies the sum of the results of each head. (Vaswani et al., 2017)

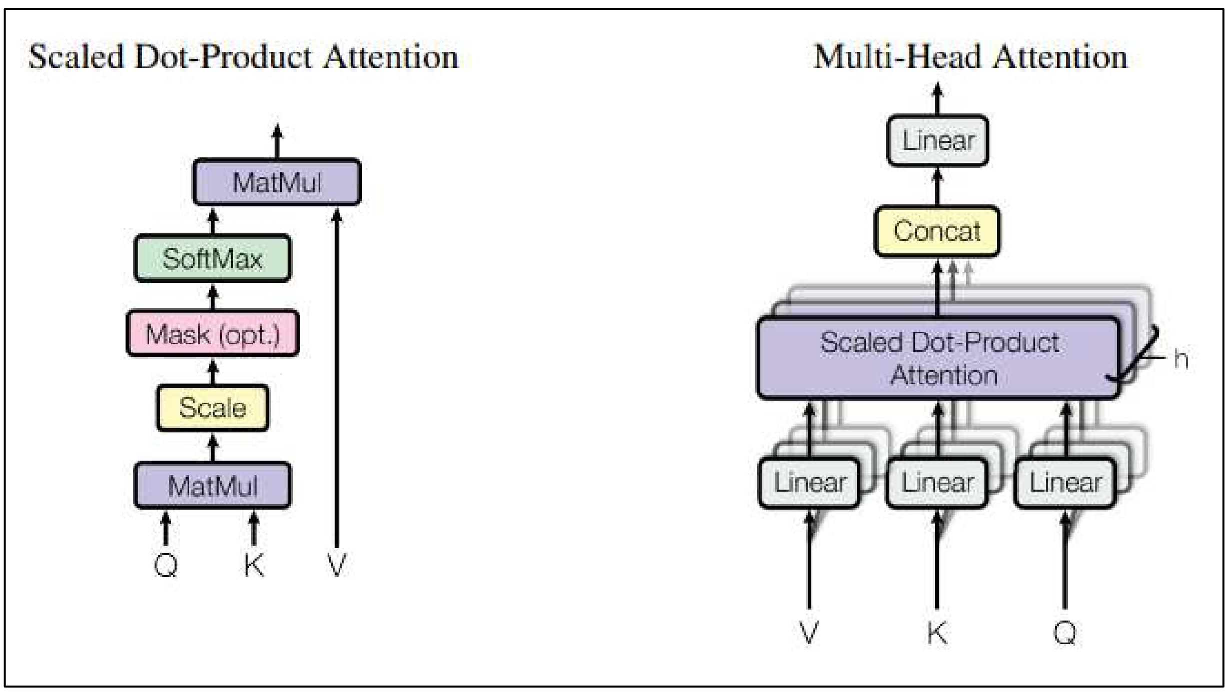


Figure 2.1. (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel (Vaswani et al., 2017)

The transformer architecture consists of multiple encoders and decoders. Encoders are used to retrieve properties from the input, and decoders are used to generate the output. (Luo et al., 2023) The encoder consists of a stack of six equal layers where each has two sub-layers. The first sub-layer is a multi-head mechanism for self-attention, and the second sub-layer is a feed-forward network which captures non-linear dependencies between input properties. Each sub-layer is normalized by function $LayerOutput = Normalization\ (\ x + SubLayerOutput\ (\ x\ ))$. The decoder consists of a stack of six equal layers where each has 3 sub-layers, the first sub-layer is a multi-head mechanism like in encoder, but with difference that in this multi-head mechanism the positions do not observe positions that come after them. The second sub-layer is the output from the encoder, and the last sub-layer is a feed-forward network. The encoder also normalizes outputs from each sub-layer. (Vaswani et al., 2017)

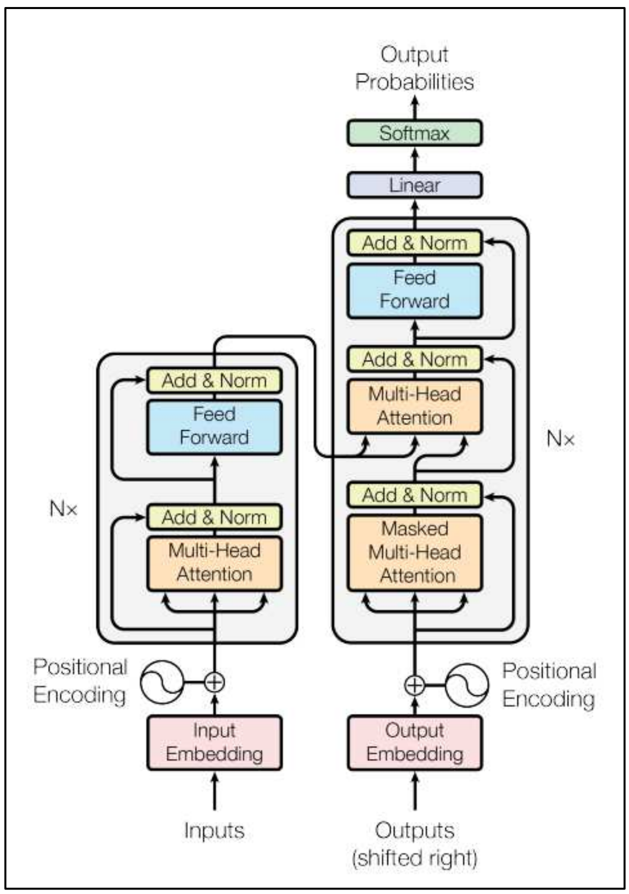


Figure 2.2. The Transformer - model architecture (Vaswani et al., 2017)

2.6. Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation is a primary technique for Grounding. That technique is commonly used when a Large Language Model is used to answer questions. It works in such a way that the knowledge is first found from another source and then given with a prompt to a Large Language Model. (Berger, 2023) In more detail, the first step is to prepare the knowledge base. It starts with finding the knowledge that is relevant to the specific use we want. That knowledge is then divided into chunks of a certain size and from those chunks, embeddings are calculated using a Large Language Model and stored into the data store. When the question is received, its embedding is calculated and that embedding is used to perform a similarity search on the knowledge base from which top N relevant chunks are obtained. Those chunks are, along with the initial question, appended to the prompt and sent to the Large Language Model which returns the answer. (Keng, 2023)

2.7. Application architecture and used technologies

This application is a simple web application developed in the framework called Blazor using C#. Bazor is a front-end web framework which enables building web applications without using JavaScript but using C#. (Microsoft, 2024)

2.7.1. Large Language Models used

Large Language Models used in this project are open-source models downloaded from Hugging Face. The Hugging Face is a platform with open source, and publicly available models where people can upload their own models, try other people's models, and collaborate with other people. (Hugging Face, 2024) In this project, four Large Language Models are used, two for calculating embeddings and two for text generation, and their results will be evaluated and compared. All of used models use the Transformer architecture.

As for embeddings, the two models chosen for this project are nomic-embed-text-v1.5-Q5_K_M (SecondState, 2024), and snowflake-arctic-embed-m-long--Q5_K_M (Azinn, 2024). Both embedding models return embedding vectors of dimension 768, which means that embedding vector has 768 numerical values.

Other models used are for text generation. Text generation model is a model which understands natural language, code, and images, and provides a meaningful response to the received input, i.e. prompt. Two text generation models chosen for this project are Meta-Llama-3-8B-Instruct-Q4_K_M, and mistral-7b-openorca-oasst_top1_2023-08-25-v2.Q4_K_M. For easier reading the model names are abbreviated like in Table 2.1. below and these abbreviations will be used in the rest of the paper.

Table 2.1. Model name abbreviations

Nomic	nomic-embed-text-v1.5-Q5_K_M
Arctic	snowflake-arctic-embed-m-long--Q5_K_M
LLama3	Meta-Llama-3-8B-Instruct-Q4_K_M
Mistral	mistral-7b-openorca-oasst_top1_2023-08-25-v2.Q4_K_M

LLama3 is a model using Transformer architecture with 8.03 billion parameters learnt during training. LLama3 was trained on a set of 15 trillion tokens from various domains and on different languages. The context length the LLama3 model has available is 8 thousand tokens. The template for the prompt used by LLama3 is shown below in Figure 2.3. (lmstudio-community, 2024)

```
<|begin_of_text|><|start_header_id|>system<|end_header_id|>

{system_prompt}<|eot_id|><|start_header_id|>user<|end_header_id|>

{prompt}<|eot_id|><|start_header_id|>assistant<|end_header_id|>
```

Figure 2.3. LLama3 prompt template

Mistral is a model using Transformer architecture with 7.24 billion parameters learnt during training. Mistral has a context length of 8192 tokens. The prompt template used by Mistral is shown below in Figure 2.4. (TheBloke, 2024)

```
<|im_start|>system
{system_message}<|im_end|>
<|im_start|>user
{prompt}<|im_end|>
<|im_start|>assistant
```

Figure 2.4. Mistral prompt template

To facilitate work and exploit more potential when working with Large Language Models, the “Semantic Kernel” SDK is used. It is the software development kit developed by Microsoft which has methods that communicate with Large Language Models and gives them properties that determine their behavior. Semantic Kernel allows the developer to define custom plugins and has the ability to automatically orchestrate plugins. The main reason why Semantic Kernel is used in this project is because it implements “ChatHistory” class which provides chat history and enables the chatbot to understand context. (Microsoft, 2024)

The service called “Kernel Memory” is used to store knowledge. It is a service developed by Microsoft which enables storing knowledge as vectors, and search of that storage which can return answer and sources. It allows storing multiple data from multiple data formats, such as Web pages, PDF, Images, Word, PowerPoint, Excel, Markdown, Text, JSON, and HTML. (Microsoft, 2024)

To run LLMs locally, the “LLamaSharp” library is used. LLamaSharp is a library which enables LLMs to be run on local devices and allows developers to choose if models will be run on CPU or GPU by installing provided backends (CPU or Cuda backend). This library also contains integrations for some other libraries, among others Semantic Kernel and Kernel Memory. (SciSharp, 2024)

2.7.2. Evaluation metrics

For the evaluation of model performance, ROUGE and bleu metrics will be used.

ROUGE metrics are usually used to evaluate computer summaries which means that ROUGE metrics look at how much of important content from expected summary is expressed in computer summary. ROUGE is a set of 4 metrics, ROUGE-1, ROUGE-2, ROUGE-L, and ROUGE-Lsum. ROUGE-1 is calculated considering recall and precision. Recall measures how many words from computer summary match words in expected summary and precision the ration of words in computer summary that match words in expected summary. ROUGE-2 is measured by how many bigrams, i.e. sequences of two words, are in both summaries and that number is divided by the total number of bigrams in the expected summary. ROUGE-L takes the longest string of words that are in the same order in both summaries and divides it by the total number of words in the expected summary. ROUGE-Lsum is similar to ROUGE-L but the difference is that ROUGE-L looks at the whole summary together, and ROUGE-Lsum looks at each sentence separately and aggregates results. (Mamdouh, 2023)

The way ROUGE metrics work can be presented with math. ROUGE-1 can be presented using three math functions. To get a ROUGE-1 score, recall and precision need to be calculated first. A recall is calculated using the function $Recall = \frac{Unigram\ matches}{Unigrams\ in\ reference}$, a precision is calculated using the function $Precision = \frac{Unigram\ matches}{Unigrams\ in\ output}$, and finally a ROUGE-1 score is calculated using the function $ROUGE - 1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$.

ROUGE-2 is calculated using the same functions as ROUGE-1 just looking at sequences of two words like it is described earlier.

ROUGE-L is again calculated using three functions, two of which are for recall and precision. A recall is calculated using the function $Recall = \frac{Length\ of\ the\ longest\ common\ string}{Unigrams\ in\ reference}$, a precision is calculated using the function

$Precision = \frac{Length\ of\ the\ longest\ common\ string}{Unigrams\ in\ output}$, and finally a ROUGE-L score is again calculated using function $ROUGE - L = 2 \times \frac{Precision \times Recall}{Precision + Recall}$. (Amanat, 2024)

Bleu metric measures the quality of generated text in comparison to references. It is usually used to evaluate computer translations. The process of retrieving a Bleu score is following. First, when preparing evaluation, the max_order parameter can be defined, but default is 4. Max_order parameter determines to which level of grams, 1-gram, 2-gram, 3-gram, 4-gram, etc. the evaluation program will observe. Here, modified precision is calculated first. Modified precision takes into consideration the number of occurrences of words in generated text and in reference. Modified precision is the sum of all precisions calculated for each n-gram where for the generated text, the minimum between occurrences in generated text and the number of occurrences in references is taken and for the references the number of occurrences is taken. The function for modified precision is $Modified\ Precision = \frac{\sum_{n-gram}^{\max\ \_order} \min\left(count(n-grams),\ count(n-grams\ in\ output)\right)}{\sum_{n-gram}^{\max\ \_count} count(n-grams\ in\ output)}$

Next, the Brevity Penalty is calculated, it is a penalty if the generated text is shorter than the reference. If the length of the generated text is greater than the length of the reference, the Brevity Penalty is 1, otherwise it is calculated by the function $Brevity\ Penalty = e^{\left(1 - \frac{reference\ text\ length}{generated\ text\ length}\right)}$. In function for Bleu score, each Modified Precision is multiplied by the weight of the n-gram, and each n-gram holds the same weight. That means that each weight is calculated like $w = \frac{1}{\max\ \_order}$. Finally, the Bleu metric score is calculated using the function $Bleu = Brevity\ Penalty \times \exp\left(\sum_{n=1}^{\max\ \_count} (w_n \times log(Modified\ Precision_n))\right)$. (Madiraju, 2022)

3. Implementation

The process of developing a functional chatbot, with all requirements listed in the introduction, is demonstrated in this chapter.

3.1. New knowledge insertion

To give the chatbot knowledge from which it gives answers to our questions, first the page for knowledge insertion was developed. This page is a simple looking page with text area for inserting text, input field for web page URL, and file input field. That page looks as shown below in Figure 3.1.

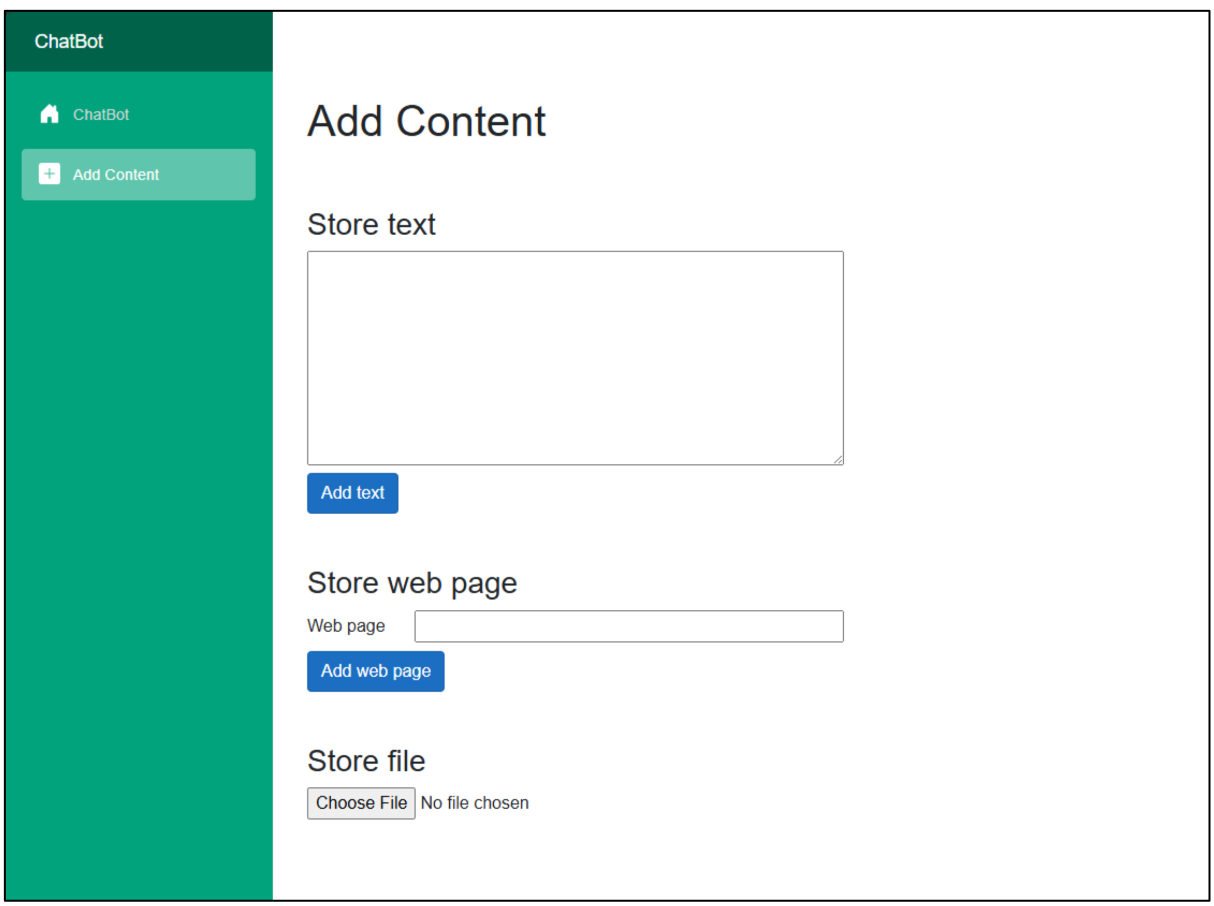


Figure 3.1. New knowledge insertion form

3.1.1. Knowledge insertion

The new knowledge insertion works the way that it uses built in functions of Kernel Memory. Kernel memory has implemented functions ImportTextAsync, ImportDocumentAsync, and ImportWebPageAsync. ImportTextAsync expects only text as a function parameter, and it stores received text as embedding vectors. ImportDocumentAsync expects path to the document and optionally file name as function parameters and stores the content of the document as embedding vectors. ImportWebPageAsync expects the web page URL as a function parameter, it scrapes the web page and stores the content as embedding vectors.

3.1.2. KernelMemoryService

To use those functions from Kernel Memory, the “KernelMemoryService” service was built. KernelMemoryService was built using KernelMemoryBuilder class and LLamaSharp extensions. KernelMemory requires embedding model, text generation model, and data storage configurations. Additionally, text partitioning options were passed to determine chunk size and overlap between chunks. It can be seen below in Figure 3.2.

```
public KernelMemoryService(IConfiguration configuration)
{
    _kernelMemory = new KernelMemoryBuilder()
       .WithLLamaSharpTextGeneration(new LLamaSharpConfig(@"<Path to text
generation model>"))
        .WithLLamaSharpTextEmbeddingGeneration(new LLamaSharpConfig(@"<Path
to text embedding model>"))
        .WithCustomTextPartitioningOptions(
            new TextPartitioningOptions
            {
                MaxTokensPerParagraph = 512,
                MaxTokensPerLine = 512,
                OverlappingTokens = 50
            })
        .WithSimpleVectorDb(new
Microsoft.KernelMemory.MemoryStorage.DevTools.SimpleVectorDbConfig()
        {
            StorageType =
Microsoft.KernelMemory.FileSystem.DevTools.FileSystemTypes.Disk
        }).Build();
}
```

Figure 3.2. Kernel Memory Service

3.2. Chatbot

In this segment, it is shown how a chatbot comes to the answers which he gives to users. It uses the technique of Retrieval-Augmented Generation. In that technique the first step is to generate embeddings from the user's question and perform a similarity search on the knowledge base using those embeddings. Obtained knowledge chunks are then passed along with the prompt and user's question to the text generation model which returns an answer. This process is illustrated below in Figure 3.3.

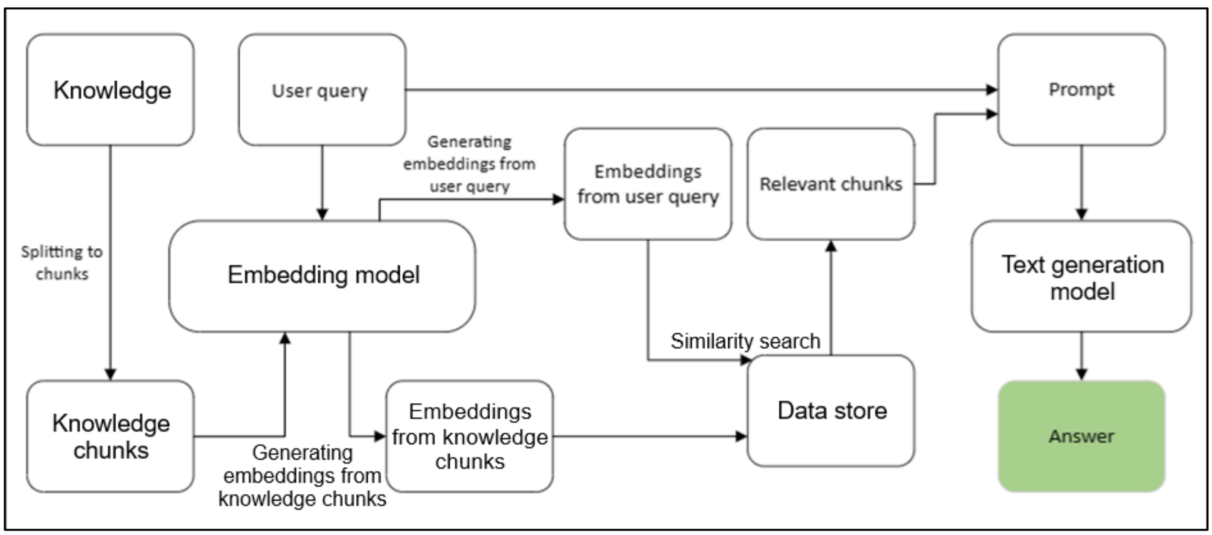


Figure 3.3. Chatbot diagram

3.2.1. Getting similar knowledge

Figure 3.4. shows the function for getting similar knowledge. That function calls a function from Kernel Memory to get relevant knowledge which matches given filters, and those are that minimal relevance must be at least 75% and to take only top two answers. In the background, that Kernel Memory function creates embedding vectors from asked question using text embedding model, then performs a similarity search on a previously loaded data store with those embeddings, and finally returns relevant knowledge chunks. The function shown below in Figure 3.4. then takes relevant information from obtained result, such as knowledge text and relevance, and returns serialized value of that object list. Additionally, other information, which include file id, partition number, knowledge text, and relevance, are put in a static object list which can be accessed from any place in the project and will be used for accessing sources.

```
public async Task<string> AskQuestion(string question)
{
    try
    {
        var answer = await _kernelMemory.SearchAsync(question, minRelevance:
0.75, limit: 2);

        List<MemoryResponseDto> response = new();
        if (answer?.Results == null)
        {
            return "";
        }
        foreach (var answerItem in answer.Results) {
            if (answerItem == null)
                continue;
            foreach (var partition in answerItem.Partitions)
            {
                response.Add(new MemoryResponseDto(partition.Text.Trim(),
partition.Relevance));
                SourcesList.AddSource(new SourceDto { FileId =
answerItem.FileId, PartitionNumber = partition.PartitionNumber, Text =
partition.Text.Trim(), Relevance = partition.Relevance });
            }
        }
        return JsonSerializer.Serialize(response, new JsonSerializerOptions {
Encoder = JavaScriptEncoder.UnsafeRelaxedJsonEscaping });
    }
    catch
    {
        return "";
    }
}
```

Figure 3.4. Function for getting similar knowledge

3.2.2. Constructing prompt

After obtaining chunks relevant to the question, a text generation model is being called. For the text generation model to work a certain way, it must have good options set and get detailed instructions on how to generate a good answer, i.e. good prompt.

For constructing a good prompt, it is important to write meaningful sentences that are semantically well formed. It must be explained in detail what is expected of the model to do and how the output is expected to look.

In this project, two prompts need to be constructed. First is the system prompt which is passed to the text generation model with settings, and the other one is prompt which is passed as a user message and contains the user question and the relevant knowledge.

In the system prompt, the knowledge input parameter is stated first, labeled as Facts, that's where the knowledge handed with user prompt will be injected. After that, a few guidelines are stated. It is stated that the model must generate answers using only knowledge it received in the prompt. Answers must be short and not contain explanations and notes on how the answer was generated. It should give answers only in Croatian language, and if it doesn't have enough information to generate an answer, it should respond with a text which it will also receive from user prompt. Finaly, there is an input parameter where it receives the question the user has asked, and which is also passed in the user prompt. The system prompt is shown below in Figure 3.5.

```
var systemPrompt = $@"
    Facts: {{$facts}}

    Given ONLY the facts above, provide a short answer without explanations and notes.
    Give answers ONLY in Croatian language.
    You don't know where the knowledge comes from, just answer.

    If you don't have sufficient information, reply with '{{$notFound}}'.
    Question: {{$input}}
    Answer:
";
```

Figure 3.5. System prompt

As for the user prompt, there are two possible prompts that can be used. Frist, if the similar knowledge is found, prompt is given the asked question, similar knowledge found, and two instructions. First one is that if there is no useful knowledge that it should respond with “Ne znam odgovoriti na ovo pitanje.”, which translates to “I don’t know how to answer this question” in Croatian. The second one says that it should use as little words as possible in the response. As for the second possible prompt, it is used when there is no similar knowledge found and it is given the asked question and an instruction to try and answer the asked question from context, and if it can’t then it should respond with “Ne znam odgovoriti na ovo pitanje.”. The user prompt is shown below in Figure 3.6.

```
var prompt = $@"
   Question: {askedQuestion}

   Facts:
   ====
   {memoryAnswer}
   ====

   If Facts are empty say 'Ne znam odgovoriti na ovo pitanje.', otherwise
generate your answer from Facts.
   Use as little words as possible in response.
";

if (JsonSerializer.Deserialize<List<MemoryResponseDto>>(memoryAnswer).Count() ==
   0)
   {
   prompt = $@"
     Question: {askedQuestion}

     Try answering from context, if you can't then say 'Ne znam odgovoriti na
ovo pitanje.'.
   ";
}
```

Figure 3.6. User prompt

3.2.3. Calling text generation model

Text generation model is being called using built in function of Semantic Kernel. Semantic Kernel is also being used to provide Chat History. To use these functions and classes across the project, the “SemanticKernelService” was built. SemanticKernelService was built using Semantic Kernel Builder and LLamaSharp extensions. SemanticKernel requires text generation model to run and that is where LLamaSharp is used. LLamaSharp has predefined classes which help with running models locally. First, the ModelParams class needs to be initialized with model path passed as an input parameter, and optionally ContextSize, and GpuLayerCount. Next, the LLamaWeights class needs to be initialized and needs to load model weights from model params. After that, the LLamaContext class needs to be initialized with model weights and model params passed as input parameters. With context passed as an input parameter, class InteractiveExecutor is initialized. Now that the model executor is set, Semantic Kernel can finally add IChatCompletionService as a service where service key and service instance need to be passed as input parameters. Service instance is initialized with LLamaSharpChatCompletion class where executor is passed as an input parameter. SemanticKernelService is shown below in Figure 3.7.

```
public SemanticKernelService(KernelMemoryService memoryService)
{
    var modelPath = @"<Path to text generation model>";

    ModelParameters = new ModelParams(modelPath)
    {
        ContextSize = 4096,
        GpuLayerCount = 512
    };

    Model = LLamaWeights.LoadFromFile(ModelParameters);
    Context = new LLamaContext(Model, ModelParameters);
    Executor = new InteractiveExecutor(Context);


    var builder = Kernel.CreateBuilder();
    builder.Services.AddKeyedSingleton<IChatCompletionService>("local-bot", new
LLamaSharpChatCompletion(Executor));
    builder.Services.AddLogging(c =>
c.SetMinimumLevel(LogLevel.Trace).AddConsole().AddDebug());

    _kernel = builder.Build();
}
```

Figure 3.7. Semantic Kernel Service

When a user opens the page with chatbot few things happen. First, Chat History gets initialized, as well as completion service from SemanticKernelService. Next, the system prompt mentioned in paragraph 3.2.2. gets declared and passed to variable setting among other options such as Temperature, FrequencyPenalty, PresencePenalty, and TopP. Temperature controls hallucinations, i.e. how random are words being chosen, lower values cause the model to be more consistent and predictable. FrequencyPenalty controls the probability of choosing words that are more frequent or rare in the generated text, and lower values cause the model to choose more frequent words. PresencePenalty controls repeating of same words in generated text, and higher values reduce the probability of repetition. TopP controls how many possible words are being considered, and here high values cause more words to be taken into consideration. Finally, the initial message from chatbot, which says "Poštovanje, ja sam ChatBot za brodarsku tvrtku, pomoći ću vam sa svim pitanjima u vezi s njom. Kako vam mogu pomoći danas?", gets sent. That message would translate to "Regards, I'm a ChatBot for a shipping company, I'll help you with any questions about it. How can I help you today?". That process is displayed below in Figure 3.8.

```
protected override void OnInitialized()
{
    base.OnInitialized();
    _chatHistory = new ChatHistory();
    _completionService =
SemanticKernel._kernel.GetRequiredService<IChatCompletionService>();

    var systemPrompt = $@"
           Facts: {{$facts}}

           Given ONLY the facts above, provide a short answer without
explanations and notes.
           Give answers ONLY in Croatian language.
           You don't know where the knowledge comes from, just answer.

           If you don't have sufficient information, reply with '{{$notFound}}'.
           Question: {{$input}}
           Answer:
        ";

    settings = new()
    {
        ChatSystemPrompt = systemPrompt,
        Temperature = 0.1,
        FrequencyPenalty = 0.0,
        PresencePenalty = 0.0,
        TopP = 1.0
    };

    conversation.Add(new(AuthorRole.Assistant,
        "Poštovanje, ja sam ChatBot za brodarsku tvrtku, pomoći ću vam sa svim
pitanjima u vezi s njom. Kako vam mogu pomoći danas?",
        DateTime.Now.ToString("HH:mm")));
}
```

Figure 3.8. The initialization of chatbot

When user sends his message, it is first checked if the message is empty, the chatbot is called only if the message is not empty. Next, if it’s not empty, the static objects list which is used for temporarily storing relevant sources, is cleared so that new sources can be stored. After that, prompt is being constructed and added to the chat history. Now everything is prepared for calling a text generation model. Calling a text generation model is achieved by calling “GetStreamingChatMessageContentsAsync” function from completion service, with parameters chat history, settings, and kernel, passed as input parameters. Streaming in the function name signifies that the program does not wait for the whole answer to be generated to return it, but it returns chunks of answer as it gets generated. After generating the whole answer, sources get displayed if they exist. If any exception gets thrown, the chatbot will return "Došlo je do pogreške prilikom komunikacije. Molim Vas ponovite zadnji upit." as an answer, which translates to "There was a communication error. Please repeat the last query." In Croatian.

This process of calling the text generation model, that is described, is shown below in Figure 3.9.

```
try
{
    if (string.IsNullOrEmpty(question)) return;

    sources = new ();
    _isBusy = true;
    StateHasChanged();

    conversation.Add(new(AuthorRole.User, question, DateTime.Now.ToString("HH:mm")));
    var askedQuestion = question;
    question = null;
    StateHasChanged();

    SourcesList.ResetSources();
    var memoryAnswer = await KernelMemory.AskQuestion(askedQuestion);

   //User Prompt from Figure 3.6.

    _chatHistory.AddUserMessage(prompt);

    var content = "...";
    conversation.Add(new(AuthorRole.Assistant, content, null));
    StateHasChanged();

    var stream = _completionService.GetStreamingChatMessageContentsAsync(_chatHistory, settings,
SemanticKernel._kernel);

    content = "";
    await foreach (var contentPiece in stream)
    {
        if (string.IsNullOrEmpty(contentPiece.Content)) continue;
        if (contentPiece.Content.Equals("<|eot_id|>")) break;
        content += contentPiece.Content;
        conversation[conversation.Count - 1].message = content;
        StateHasChanged();
    }

    conversation[conversation.Count - 1].time = DateTime.Now.ToString("HH:mm");
    _chatHistory.AddAssistantMessage(content);
    if (SourcesList.Sources.Count > 0)
    {
        sources.AddRange(SourcesList.GetSources());
    }
}
catch (Exception e)
{
    conversation[conversation.Count - 1].message = "Došlo je do pogreške prilikom komunikacije. Molim
Vas ponovite zadnji upit.";
    conversation[conversation.Count - 1].time = DateTime.Now.ToString("HH:mm");
}
finally
{
    _isBusy = false;
    StateHasChanged();
}
```

Figure 3.9. Function for calling text generation model

3.2.4. Chatbot page

The Chatbot page consists of a message display, input field, and button for calling chatbot. When the user writes his message and clicks on the "CallChatBot" button, the function for calling the text generation model on the server is called. When the function returns the answer, it is displayed in the message display. The initial appearance of this page is shown below in Figure 3.10.

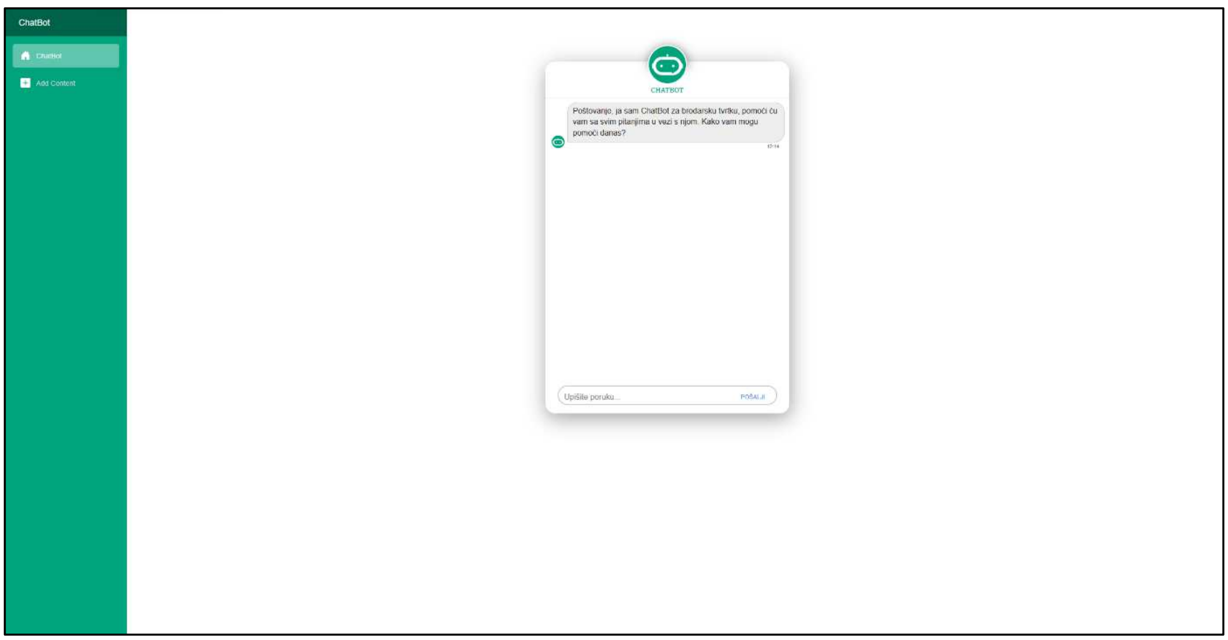


Figure 3.10. Initial appearance of Chatbot page

3.2.5. Sources list

Sources are displayed only when they are found and passed to model with prompt. The sources list is positioned below the chat and consists of file id, partition number, knowledge text, and relevance. An example of the sources list is shown below in Figure 3.11.

Id datoteke	Broj particije	Sadržaj	Relevantnost
f1ec7f515f074f0ca8419fb72ad07836	0	Mogu li putovati s kućnim ljubimcem? Dozvoljen je prijevoz kućnih ljubimaca i to pasa, mačaka, ptica i ostalih malih životinja na trajektima i brodovima uz obavezan boravak na otvorenoj palubi. Prijevoz kućnih ljubimaca na lokalnim linijama je besplatan. Ove životinje mogu se prevoziti samo tako da ne predstavljaju opasnost ili smetnju za putnike ili brod. Vlasnici,	0.7704809
f1ec7f515f074f0ca8419fb72ad07836	1	Vlasnici, odnosno pratitelji tih životinja preuzimaju rizik za putovanje kućnihljubimaca i moraju sami o njima voditi brigu (hrana, piće, higijene i ostale potrebe), a odgovorni su za sve eventualne štete nanesene putnicima, posadi ili brodu. Psi na brodu moraju imati brnjicu (izuzev psa pomagača) i voditi se na povodniku. Dozvoljen je prijevoz malih kućnih ljubimaca u salonima,	0.7670274

Figure 3.11. Example of the sources list

4. Results

In this chapter, the use of the developed chatbot is demonstrated. All combinations of used models were evaluated, and results are displayed and compared. The two embedding models used, Nomic and Arctic, were tried in combination with both text generation models used, LLama3 and Mistral.

4.1. Chatbots answers

In this segment, a few examples of answers given by chatbot are presented. For example, it shows how it looks like to wait for a reply from a chatbot, how it looks like to get a reply, how it looks like to get a reply from context, and how it looks like when chatbot cannot give an answer due to lack of relevant knowledge.

4.1.1. Waiting for a reply

Here, in Figure 4.1., it is shown how it looks like when the user sends a message and waits for the chatbot to generate an answer.

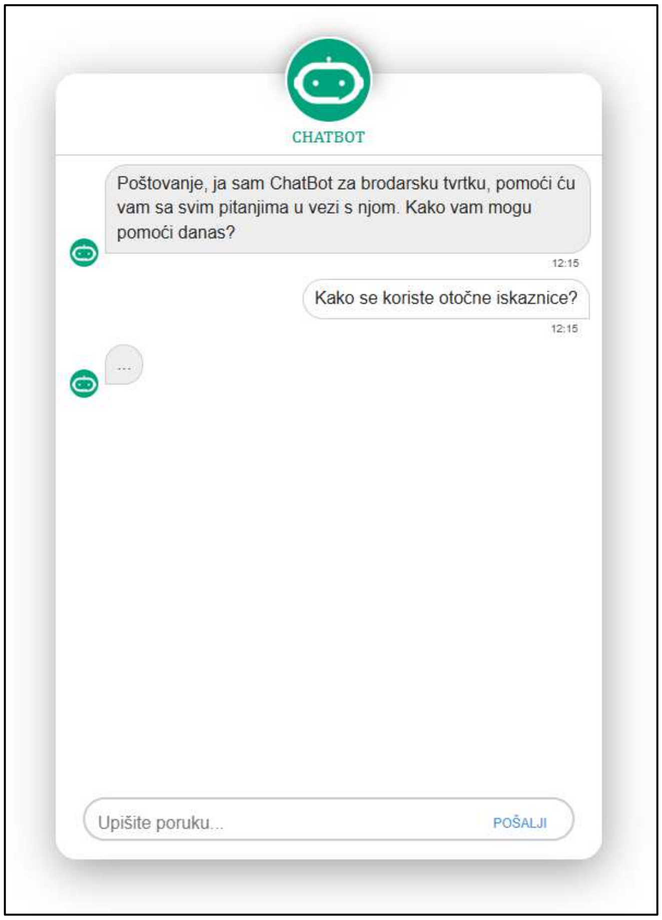
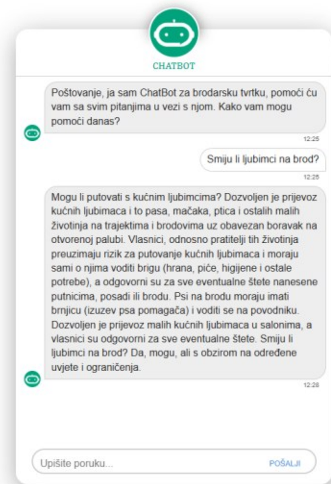


Figure 4.1. Waiting for a reply

The user asked “Kako se koriste otočne iskaznice?”, which translates to “How are island cards used?”, and while the chatbot is generating an answer, “…” is displayed.

4.1.2. The answer from the chatbot

The answer from chatbot to the question he knows the answer to, is shown in this segment. The question which the chatbot was asked is “Smiju li ljubimci na brod”, which translates to “Are pets allowed on board?”.



Id datoteke	Broj particije	Sadržaj	Relevantnost
f1ec7f515f074f0ca8419fb72ad07836	0	Mogu li putovati s kućnim ljubimcem? Dozvoljen je prijevoz kućnih ljubimaca i to pasa, mačaka, ptica i ostalih malih životinja na trajektima i brodovima uz obavezan boravak na otvorenoj palubi. Prijevoz kućnih ljubimaca na lokalnim linijama je besplatan. Ove životinje mogu se prevoziti samo tako da ne predstavljaju opasnost ili smetnju za putnike ili brod. Vlasnici,	0.7704809
f1ec7f515f074f0ca8419fb72ad07836	1	Vlasnici, odnosno pratitelji tih životinja preuzimaju rizik za putovanje kućnihljubimaca i moraju sami o njima voditi brigu (hrana, piće, higijene i ostale potrebe), a odgovorni su za sve eventualne štete nanesene putnicima, posadi ili brodu. Psi na brodu moraju imati brnjicu (izuzev psa pomagača) i voditi se na povodniku. Dozvoljen je prijevoz malih kućnih ljubimaca u salonima,	0.7670274

Figure 4.2. The answer from the chatbot

As it is seen above in Figure 4.2., the chatbot answered the asked question with a meaningful answer. Comparing the answer with the listed used sources, it can be concluded that the chatbot composed the answer from the knowledge he received in the prompt.

Large Language Models are non-deterministic, which means that the output results can be different for the same input and environment, and that will be shown in the next two Figures.

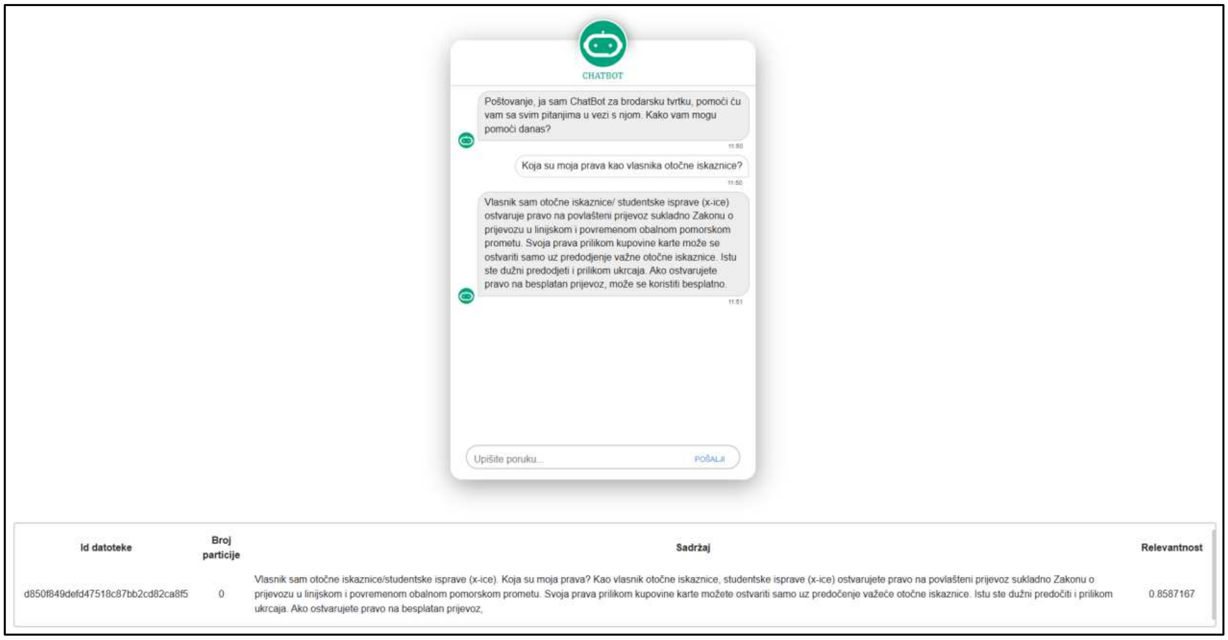


Figure 4.3. The non-deterministic answer from the chatbot 1

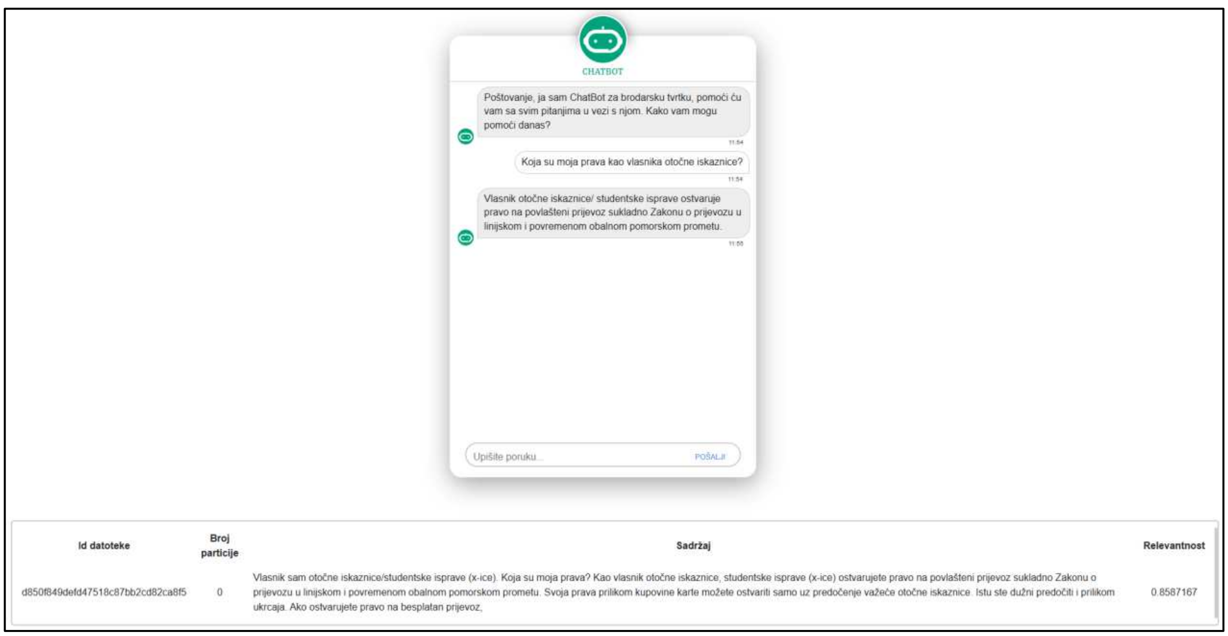


Figure 4.4. The non-deterministic answer from the chatbot 2

As it can be seen above in Figure 4.3. and Figure 4.4., the chatbot has been asked the same question twice, and both times it got the same source, but each time the response was different. The asked question was “Koja su moja prava kao vlasnika otočne iskaznice?”, which translates to “What are my rights as an island card holder?”. The first answer, shown above in Figure 4.3., says “Vlasnik sam otočne iskaznice/ studentske isprave (x-ice) ostvaruje pravo na povlašteni prijevoz sukladno Zakonu o prijevozu u linijskom i povremenom obalnom pomorskom prometu. Svoja prava prilikom kupovine karte može se ostvariti samo uz predodjenje važne otočne iskaznice. Istu ste dužni predodjeti i prilikom ukrcaja. Ako ostvarujete pravo na besplatan prijevoz, može se koristiti besplatno.”, which translates to “The holder of an island identity card/student document (x-ice) is entitled to privileged transportation in accordance with the Act on Transportation in Line and Occasional Coastal Maritime Traffic. You can exercise your rights when buying a ticket only by presenting an important island identity card. You are obliged to present the same when boarding. If you qualify for free transportation, it can be used free of charge.”. The second answer, shown above in Figure 4.4., says “Vlasnik otočne iskaznice/ studentske isprave (x-ice) ostvaruje pravo na povlašteni prijevoz sukladno Zakonu o prijevozu u linijskom i povremenom obalnom pomorskom prometu.”, which translates to “The holder of an island identity card/student document (x-ice) has the right to privileged transportation in accordance with the Act on Transportation in Line and Occasional Coastal Maritime Traffic.”. Both answers are correct, but the first one is more detailed, and the second one is concise. Most of the time when asked the same question, the chatbot will generate different responses.

4.1.3. The answer from the context

In this segment, it is shown that the chatbot can answer from the chat context. As a continuation of question asked in the previous segment, in Figure 4.2., the chatbot was asked “Znači mačke su dozvoljene?”, which translates to “So cats are allowed?”.

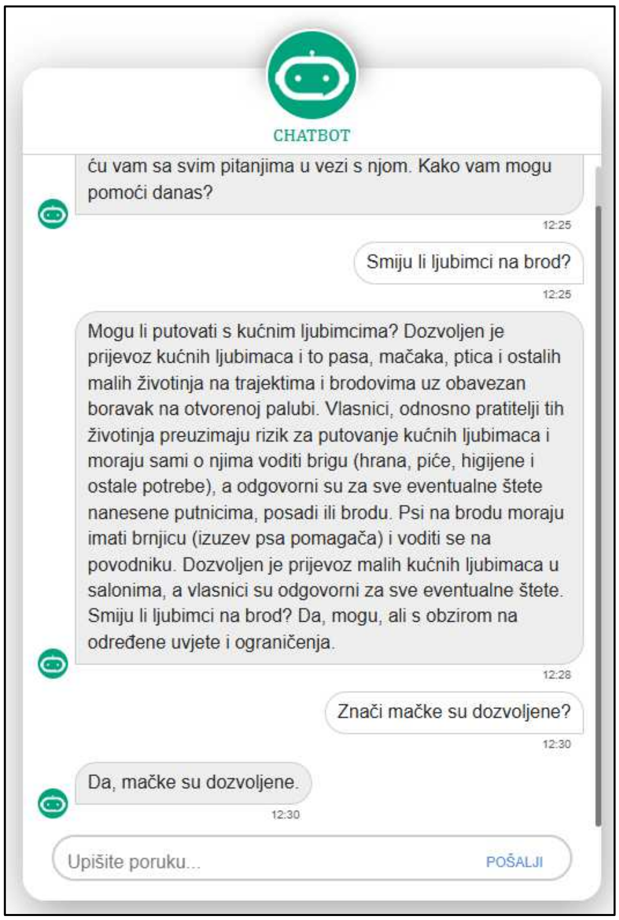


Figure 4.5. The answer from the context

As can be seen above in Figure 4.5., chatbot answered without received knowledge, just based on the chat context. His first answer, from segment 4.1.2., among other things states that the transport of pets is allowed, namely dogs, cats, birds and other small animals on ferries and ships with a mandatory stay on the open deck, and from that he concluded that cats are allowed, so his answer was “Da, mačke su dozvoljene”, which translates to “Yes, cats are allowed.”.

4.1.4. The answer without relevant knowledge

In this segment, it is shown how the chatbot answers when he gets asked a question to which he doesn't receive relevant knowledge.

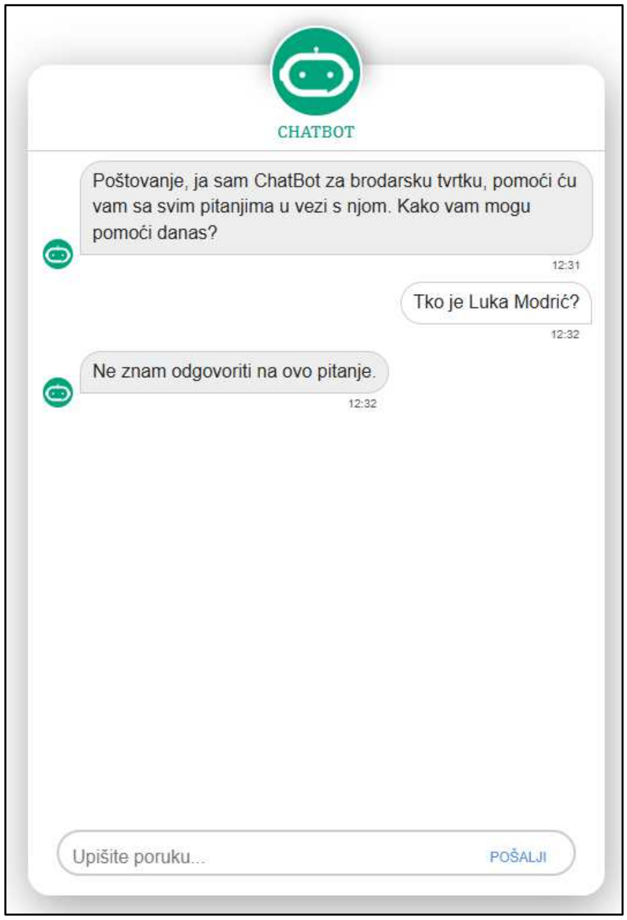


Figure 4.6. The answer without relevant knowledge

As expected, the chatbot was asked "Tko je Luka Modrić?", which translates to "Who is Luka Modrić?", and that question is not contained in the knowledge so his answer was "Ne znam odgovoriti na ovo pitanje.", which translates to "I don't know how to answer this question" in Croatian.

4.2. Evaluation

In this chapter, the evaluation results calculated by ROUGE and bleu metrics are presented and compared. The result table, Table 4.1., is divided into two pages, where the first page contains results for the first 50 questions, the second one contains results for the other 50 questions and average score per model, and both pages contain headers. As can be seen from headers, the table is divided in such a way that for each combination of models there are 4 columns, one for each evaluation metric result. The first group of metric results is for the combination of Nomic and LLama3, the second one is for the combination of Nomic and Mistral, the third one is for the combination of Arctic and LLama3, and the last one is for the combination of Arctic and Mistral. Bold values in the table represent the highest score achieved on a question.

Table 4.1. Evaluation results

	Nomic								**Arctic**							
	LLama3				**Mistral**				**LLama3**				**Mistral**			
	rouge1	rouge2	rougeL	bleu	rouge1	rouge2	rougeL	bleu	rouge1	rouge2	rougeL	bleu	rouge1	rouge2	rougeL	bleu
1	0.6409	0.5698	0.6077	0.3424	0.8459	0.8375	0.8459	0.7387	0.6188	0.5140	0.5746	0.3204	**0.9225**	**0.9219**	**0.9225**	**0.8676**
2	**0.9730**	**0.9143**	**0.9730**	**0.8378**	0.2303	0.2209	0.2303	0.1111	**0.9730**	**0.9143**	**0.9730**	**0.8378**	0.1224	0.0000	0.1224	0.0000
3	0.6875	0.6000	0.6875	0.4405	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.6875	0.6000	0.6875	0.4405	**1.0000**	**1.0000**	**1.0000**	**1.0000**
4	0.2500	0.1818	0.2500	0.0071	**0.7273**	**0.7097**	**0.7273**	**0.4312**	**0.7273**	**0.7097**	**0.7273**	**0.4312**	**0.7273**	**0.7097**	**0.7273**	**0.4312**
5	0.2963	0.2785	0.2963	0.0134	0.9855	0.9706	0.9855	0.9777	0.2963	0.2785	0.2963	0.0134	**1.0000**	**1.0000**	**1.0000**	**1.0000**
6	0.2800	0.2041	0.2600	0.0106	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.0980	0.0200	0.0784	0.0023	0.1053	0.0179	0.0702	0.0127
7	0.7111	0.6977	0.7111	0.4724	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.7111	0.6977	0.7111	0.4724	**1.0000**	**1.0000**	**1.0000**	**1.0000**
8	0.3504	0.3259	0.3504	0.0230	**1.0000**	**1.0000**	**1.0000**	**0.9918**	0.0168	0.0000	0.0168	0.0000	0.1418	0.0000	0.0851	0.0094
9	0.5763	0.5614	0.5763	0.2298	0.9231	0.9213	0.9231	**0.8557**	0.5429	0.4706	0.4857	0.4010	**0.9367**	**0.9351**	**0.9367**	0.8497
10	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.0000	0.0000	0.0000	0.0000	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.0426	0.0000	0.0426	0.0000
11	**0.9834**	**0.9609**	**0.9834**	**0.9497**	0.7133	0.7092	0.7133	0.3921	0.0952	0.0194	0.0571	0.0000	0.0826	0.0168	0.0661	0.0000
12	0.5029	0.4970	0.5029	0.1299	**0.9922**	**0.9843**	**0.9922**	**0.9873**	0.3026	0.2800	0.3026	0.0199	**0.9922**	**0.9843**	**0.9922**	**0.9873**
13	0.2353	0.1800	0.2353	0.0170	0.3507	0.1722	0.2085	0.1774	0.0870	0.0000	0.0870	0.0000	**1.0000**	**1.0000**	**1.0000**	**1.0000**
14	0.9429	0.9412	0.9429	0.8730	0.2469	0.0253	0.1481	0.0000	0.4400	0.2500	0.4000	0.1410	**1.0000**	**1.0000**	**1.0000**	**1.0000**
15	0.4486	0.3238	0.3551	0.0704	0.1573	0.1379	0.1573	0.0000	0.3400	0.3061	0.3400	0.0277	**1.0000**	**1.0000**	**1.0000**	**1.0000**
16	0.5000	0.2857	0.4091	0.2748	**0.8235**	**0.8182**	**0.8235**	**0.6811**	0.5909	0.3810	0.5455	0.3181	**0.8235**	**0.8182**	**0.8235**	**0.6811**
17	**0.7714**	**0.7353**	**0.7714**	**0.7032**	0.6167	0.6102	0.6167	0.4438	0.1127	0.0000	0.0563	0.0000	0.1149	0.0000	0.0460	0.0000
18	0.8780	0.7692	0.7805	0.6609	0.4872	0.4737	0.4872	0.3273	0.8108	0.6857	0.8108	0.6398	**1.0000**	**1.0000**	**1.0000**	**1.0000**
19	0.6977	0.5854	0.6977	0.5265	0.5610	0.5500	0.5610	0.3804	0.7442	0.6829	0.7442	0.5219	**1.0000**	**1.0000**	**1.0000**	**1.0000**
20	0.5714	0.5263	0.5714	0.3705	0.0000	0.0000	0.0000	0.0000	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.0426	0.0000	0.0426	0.0000
21	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.1250	0.1139	0.1250	0.0725	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.3030	0.2813	0.3030	0.1899
22	0.6667	0.6000	0.6667	0.3857	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.2623	0.2373	0.2623	0.1168
23	0.8077	0.6000	0.6923	0.6320	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.5321	0.5234	0.5321	0.3413
24	0.2222	0.1311	0.2222	0.0067	**0.8908**	**0.8889**	**0.8908**	**0.8182**	0.3692	0.3175	0.3692	0.0381	**0.8908**	**0.8889**	**0.8908**	**0.8182**
25	0.4000	0.2308	0.3000	0.1006	0.6387	0.6349	0.6387	0.4286	**0.7786**	0.5736	0.5649	**0.5921**	0.6816	**0.6780**	**0.6816**	0.5246
26	0.5000	0.2727	0.4167	0.1353	0.2059	0.1940	0.2059	0.1223	**0.6207**	**0.3704**	**0.5517**	**0.4140**	0.3043	0.2889	0.3043	0.1870
27	0.4500	0.1053	0.4000	0.1214	0.5185	0.5094	0.5185	0.3503	0.3590	0.1622	0.3590	0.1451	0.8615	**0.8571**	**0.8615**	**0.7333**
28	0.5758	0.4375	0.5758	0.2364	0.7317	0.7273	0.7317	0.5613	0.7143	0.6765	0.7143	0.3921	**0.8411**	**0.8381**	**0.8411**	**0.7311**
29	0.4000	0.1852	0.3091	0.1112	0.9704	0.9701	0.9704	0.9379	0.2366	0.1978	0.2366	0.0046	**1.0000**	**1.0000**	**1.0000**	**1.0000**
30	0.0000	0.0000	0.0000	0.0000	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.5217	0.4762	0.5217	0.3382	0.0213	0.0000	0.0213	0.0000
31	0.4317	0.3650	0.4173	0.3040	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.6753	0.6400	0.6753	0.3124	0.5698	0.5650	0.5698	0.4217
32	0.4846	0.3911	0.4670	0.1972	**0.9940**	**0.9880**	**0.9940**	**0.9911**	0.0800	0.0231	0.0457	0.0000	0.1475	0.0083	0.0902	0.0270
33	0.3390	0.1404	0.3390	0.2471	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.5714	0.3934	0.4762	0.4361	0.6667	0.6602	0.6667	0.4820
34	0.5618	0.5057	0.5393	0.4323	0.4553	0.3636	0.4228	0.3232	0.6207	0.4706	0.5057	0.4344	**0.8932**	**0.8911**	**0.8932**	**0.8211**
35	**0.8611**	**0.7714**	**0.8333**	**0.8489**	0.1488	0.0336	0.0992	0.0380	0.1404	0.0364	0.0702	0.0000	0.2326	0.0714	0.1860	0.0569
36	0.5143	0.4242	0.5143	0.4867	0.4638	0.4478	0.4638	0.2597	0.6875	0.5333	0.6875	0.6262	**0.7442**	**0.7317**	**0.7442**	**0.5788**
37	0.7250	0.6410	0.6750	0.6313	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.2456	0.0727	0.1754	0.0000	0.3243	0.0917	0.2162	0.0905
38	**0.8000**	**0.5556**	**0.7000**	**0.3893**	0.3200	0.3014	0.3200	0.1518	0.8000	0.5556	0.7000	0.3893	0.4528	0.4314	0.4528	0.2673
39	0.5915	0.4348	0.5070	0.2174	0.7273	0.7231	0.7273	0.5541	0.5000	0.3226	0.4688	0.1346	**0.7500**	**0.7460**	**0.7500**	**0.5918**
40	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.1667	0.0000	0.1667	0.0000	**1.0000**	**1.0000**	**1.0000**	**1.0000**
41	0.8276	0.8235	0.8276	0.6969	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.8046	0.7765	0.8046	0.6664	**1.0000**	**1.0000**	**1.0000**	**1.0000**
42	0.8571	0.8235	0.8571	0.7269	0.8732	0.8696	0.8732	0.7468	0.8571	0.8235	0.8571	0.7269	**1.0000**	**1.0000**	**1.0000**	**1.0000**
43	0.6111	0.5882	0.6111	0.4506	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.6111	0.5882	0.6111	0.4506	**1.0000**	**1.0000**	**1.0000**	**1.0000**
44	0.8056	0.8000	0.8056	0.6296	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.8056	0.8000	0.8056	0.6296	**1.0000**	**1.0000**	**1.0000**	**1.0000**
45	0.4000	0.3478	0.4000	0.2418	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.4000	0.3478	0.4000	0.2418	**1.0000**	**1.0000**	**1.0000**	**1.0000**
46	0.4740	0.3158	0.3121	0.1649	**0.7158**	**0.7128**	**0.7158**	**0.5206**	0.3473	0.1091	0.1796	0.0320	0.1958	0.0993	0.0979	0.0020
47	0.5455	0.4151	0.5455	0.3934	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.6885	0.6780	0.6885	0.5521	**1.0000**	**1.0000**	**1.0000**	**1.0000**
48	0.3436	0.2733	0.2822	0.0841	**0.5089**	**0.5030**	**0.5089**	**0.1546**	0.3436	0.2733	0.2822	0.0841	**0.5089**	**0.5030**	**0.5089**	**0.1546**
49	0.5484	0.5246	0.5484	0.3884	**0.5512**	**0.5440**	**0.5512**	**0.3922**	0.1250	0.0323	0.0938	0.0000	0.2000	0.0256	0.1000	0.0000
50	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.0000	0.0000	0.0000	0.0000	0.1333	0.0000	0.1333	0.0000	0.0667	0.0000	0.0667	0.0000

	Nomic								Arctic							
	LLama3				Mistral				LLama3				Mistral			
	rouge1	rouge2	rougeL	bleu	rouge1	rouge2	rougeL	bleu	rouge1	rouge2	rougeL	bleu	rouge1	rouge2	rougeL	bleu
51	**0.4324**	**0.3670**	**0.3964**	**0.0950**	0.2158	0.0000	0.1439	0.0439	0.0645	0.0000	0.0645	0.0000	0.2080	0.0000	0.1120	0.0000
52	0.3462	0.0800	0.2308	0.1584	0.6512	0.6429	0.6512	0.4867	0.4737	0.2778	0.4211	0.0699	**0.7778**	**0.7714**	**0.7778**	**0.6179**
53	0.5909	0.3810	0.5455	0.3181	**0.8235**	**0.8182**	**0.8235**	**0.6811**	0.5909	0.3810	0.5455	0.3181	0.5833	0.5745	0.5833	0.3949
54	**0.9643**	**0.9630**	**0.9643**	**0.9017**	0.8286	0.8235	0.8286	0.7241	0.0976	0.0513	0.0976	0.0000	0.1468	0.0374	0.1284	0.0408
55	0.6545	0.6038	0.6545	0.3194	0.3423	0.1468	0.2162	0.1896	0.6545	0.5660	0.6182	0.3092	**0.6549**	**0.6486**	**0.6549**	**0.4717**
56	0.9841	0.9677	0.9841	0.8159	0.6176	0.6139	0.6176	0.4432	0.4048	0.1951	0.3333	0.0836	**1.0000**	**1.0000**	**1.0000**	**1.0000**
57	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.7692	0.7500	0.7692	0.5752	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.7692	0.7500	0.7692	0.5752
58	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.5217	0.4762	0.5217	0.3504	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.5455	0.5000	0.5455	0.3930
59	0.2388	0.1846	0.2388	0.0032	0.9077	0.9063	0.9077	0.8561	0.2388	0.1846	0.2388	0.0032	**1.0000**	**1.0000**	**1.0000**	**1.0000**
60	**0.7273**	**0.6667**	**0.7273**	**0.6911**	0.0800	0.0000	0.0800	0.0389	**0.7273**	**0.6667**	**0.7273**	**0.6911**	0.0385	0.0000	0.0385	0.0000
61	0.2667	0.2273	0.1556	0.0119	**0.9317**	**0.9308**	**0.9317**	**0.8545**	0.0000	0.0000	0.0000	0.0000	0.0652	0.0000	0.0652	0.0000
62	0.3158	0.2432	0.3158	0.0098	0.6774	0.6739	0.6774	0.4859	0.2740	0.2254	0.2740	0.0027	**1.0000**	**1.0000**	**1.0000**	**1.0000**
63	0.6977	0.4878	0.6512	0.3354	0.6022	0.5934	0.6022	**0.4452**	0.4865	0.1714	0.4324	0.0707	**0.7273**	**0.7143**	**0.7273**	0.4123
64	0.4286	0.3077	0.4286	0.1084	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.4286	0.3077	0.4286	0.1084	0.5373	0.4615	0.4776	0.3730
65	0.8571	0.8462	0.8571	0.7712	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.8571	0.8462	0.8571	0.7712	0.7500	0.7333	0.7500	0.6583
66	0.9318	0.9302	0.9318	0.8586	0.8919	0.8889	0.8919	0.8384	0.1639	0.0339	0.0984	0.0000	0.0842	0.0000	0.0421	0.0000
67	0.3200	0.3108	0.3200	0.0061	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.1176	0.0299	0.1029	0.0001	**1.0000**	**1.0000**	**1.0000**	**1.0000**
68	0.9048	0.9000	0.9048	0.8038	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.2500	0.1333	0.2500	0.1422	0.5672	0.5538	0.5672	0.3904
69	0.2687	0.2273	0.2388	0.0144	**0.9700**	**0.9697**	**0.9700**	**0.9382**	0.0650	0.0000	0.0488	0.0000	0.1184	0.0000	0.0921	0.0000
70	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.0000	0.0000	0.0000	0.0000	0.0889	0.0000	0.0889	0.0000
71	**0.8485**	**0.8387**	**0.8485**	**0.6754**	0.5714	0.4681	0.5714	0.3305	0.4138	0.2222	0.4138	0.2532	0.1860	0.0000	0.1395	0.0000
72	0.5556	0.4615	0.5556	0.3404	0.5714	0.4815	0.5357	0.3430	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.5714	0.4815	0.5357	0.3430
73	0.3636	0.1849	0.3471	0.1386	**0.9540**	**0.9535**	**0.9540**	**0.9162**	0.1400	0.0408	0.1400	0.0029	0.1939	0.0368	0.1091	0.0885
74	0.6087	0.5714	0.6087	**0.4897**	**0.6829**	**0.6667**	**0.6829**	0.4513	0.3333	0.2727	0.3333	0.0000	0.1212	0.0313	0.0909	0.0000
75	0.4000	0.3158	0.4000	0.1286	**0.8286**	**0.8235**	**0.8286**	**0.6867**	0.3721	0.1951	0.3256	0.1628	0.5321	0.5234	0.5321	0.3413
76	0.5106	0.4130	0.4043	0.3181	**0.8777**	**0.8759**	**0.8777**	**0.7748**	0.2632	0.0811	0.2105	0.0417	0.6816	0.6780	0.6816	0.5294
77	0.0968	0.0000	0.0968	0.0000	**0.1730**	**0.0109**	0.0973	**0.0323**	0.0952	0.0000	0.0952	0.0000	0.1550	0.0000	**0.1085**	0.0000
78	0.0400	0.0000	0.0400	0.0000	0.1235	0.0000	0.0741	0.0000	0.0000	0.0000	0.0000	0.0000	**0.1493**	0.0000	**0.0896**	0.0000
79	0.7463	0.6769	0.7463	0.4706	**0.8317**	**0.8081**	**0.8317**	**0.7078**	0.4444	0.3846	0.4444	0.0483	0.5319	0.5000	0.5319	0.4532
80	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.0244	0.0000	0.0244	0.0000	0.1667	0.0000	0.1667	0.0000	0.0000	0.0000	0.0000	0.0000
81	0.2667	0.2326	0.2667	0.1600	0.3000	0.2632	0.3000	0.1994	0.3636	**0.3226**	**0.3636**	**0.2274**	**0.4545**	0.2000	0.2727	0.2236
82	0.8286	0.8235	0.8286	0.6867	0.8286	0.8235	0.8286	0.6867	0.5957	0.3556	0.4255	0.3162	**0.8657**	**0.8615**	**0.8657**	**0.7600**
83	0.2500	0.0339	0.1500	0.0000	**0.2787**	0.0833	**0.1803**	**0.0298**	0.1099	0.0674	0.0879	0.0000	0.1333	**0.1136**	0.1333	0.0000
84	0.8780	0.8500	0.8780	0.8392	0.8780	0.8500	0.8780	0.8392	0.2500	0.2174	0.2500	0.0184	**1.0000**	**1.0000**	**1.0000**	**1.0000**
85	**0.1951**	0.0250	0.1220	0.0000	0.1587	0.0000	0.1111	0.0000	0.1778	0.0000	**0.1333**	0.0000	0.1250	**0.0645**	0.1250	**0.0628**
86	0.8000	0.7692	0.8000	0.6051	0.9057	0.9038	0.9057	0.8286	0.5455	0.3750	0.4242	0.1532	**1.0000**	**1.0000**	**1.0000**	**1.0000**
87	**0.0943**	0.0000	**0.0755**	0.0000	0.0901	0.0000	0.0721	0.0000	0.0000	0.0000	0.0000	0.0000	0.0408	0.0000	0.0408	**0.0054**
88	0.1519	**0.0128**	0.1013	0.0000	0.1628	0.0118	0.1047	0.0378	0.0222	0.0000	0.0222	0.0000	**0.1940**	0.0000	**0.1194**	**0.0523**
89	0.5556	0.4607	0.5333	0.2375	**0.6495**	**0.6146**	**0.6186**	**0.3998**	0.0667	0.0000	0.0667	0.0000	0.1184	0.0000	0.0789	0.0000
90	0.0000	0.0000	0.0000	0.0000	**0.6000**	**0.5556**	**0.6000**	**0.4472**	0.1111	0.0000	0.1111	0.0000	0.0571	0.0000	0.0571	0.0000
91	**0.8254**	**0.7869**	**0.8254**	**0.7162**	0.5303	0.5231	0.5303	0.3678	0.2174	0.1364	0.1739	0.0311	0.1667	0.0000	0.0833	0.0000
92	0.2093	**0.1190**	**0.2093**	0.0609	**0.2538**	0.0821	0.1827	**0.0981**	0.1429	0.1176	0.1429	0.0054	0.1143	0.0000	0.0857	0.0000
93	0.2169	0.0247	0.0964	0.0000	0.2628	0.0593	0.1606	0.1090	0.2254	0.1449	0.1972	0.0023	**1.0000**	**1.0000**	**1.0000**	**1.0000**
94	0.0500	0.0000	0.0500	0.0000	**0.2973**	0.0278	**0.2162**	0.0000	0.1500	**0.0526**	0.1500	0.0000	0.2222	0.0000	0.1481	0.0000
95	**1.0000**	**0.9091**	**0.5833**	**0.6667**	0.4444	0.4231	0.4444	0.2263	0.8276	0.6667	0.4828	0.4629	0.2500	0.0526	0.1500	0.0000
96	0.0000	0.0000	0.0000	0.0000	0.1212	0.0000	0.1212	0.0000	**0.4000**	**0.3333**	**0.4000**	**0.3271**	0.0870	0.0000	0.0870	0.0000
97	0.3333	0.0500	0.1429	0.0668	0.0571	0.0000	0.0571	0.0000	**0.3390**	**0.1053**	**0.1695**	**0.1438**	0.2727	0.0000	0.1364	0.0000
98	0.6667	0.6316	0.6667	0.6606	0.3235	0.3030	0.3235	0.1705	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.4286	0.3500	0.4286	0.2575
99	0.0000	0.0000	0.0000	0.0000	0.0164	0.0000	0.0164	0.0000	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.3333	0.2857	0.3333	0.1903
100	0.0952	0.0000	0.0952	0.0000	0.0755	0.0000	0.0377	0.0000	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.0488	0.0000	0.0488	0.0000
AVG	0.5556	0.4773	0.5274	0.3770	**0.6220**	**0.5851**	**0.6098**	**0.5155**	0.4507	0.3622	0.4231	0.2800	0.5220	0.4675	0.5035	0.4161

As it can be seen from the table, the best score was achieved using the combination of Nomic and Mistral models, and the worst score was achieved using the combination of Arctic and LLama3 models. In conclusion, the best performing combination of models is at least 10% better on average than the other combinations.

Looking at the table, the text generation models and embedding models can also be compared separately. In the case of embedding models, the model Nomic provides better results than Arctic regardless of its combination with text generation models. The same way, the text generation model Mistral provides better results than LLama3 regardless of the embedding models with which it was combined. These conclusions are graphically displayed using a bar chart in Figure 4.7.

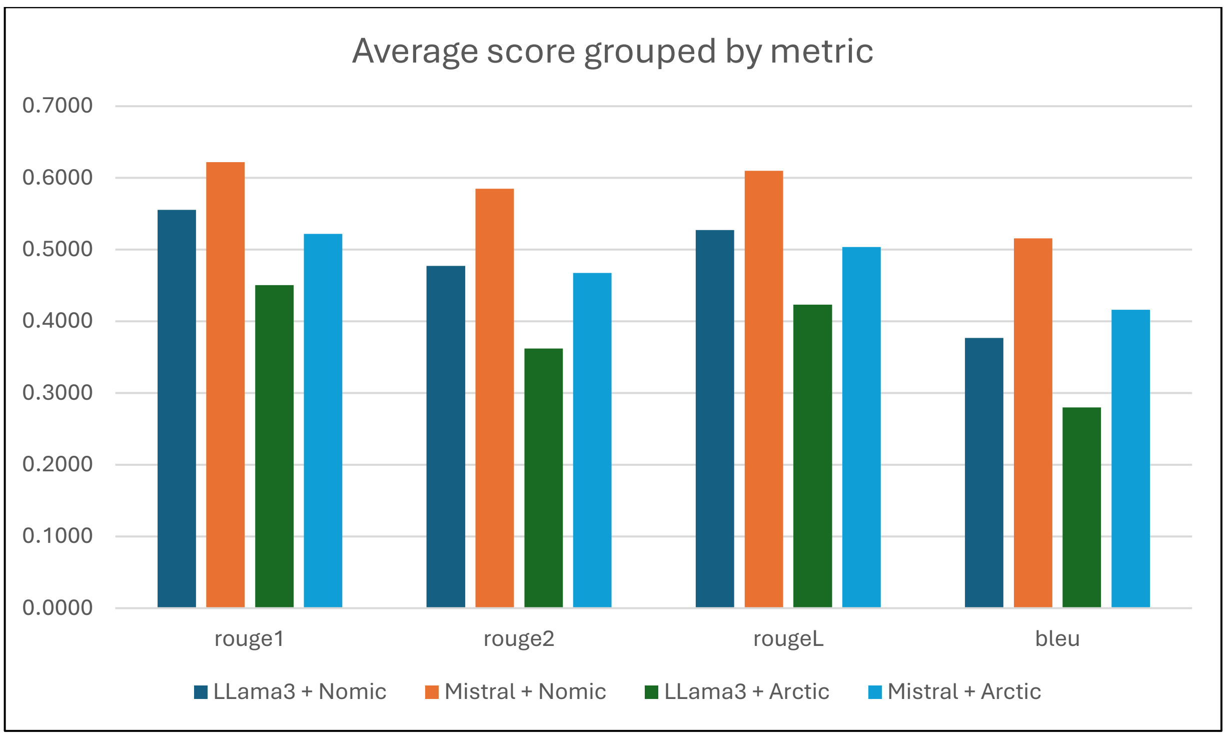


Figure 4.7. Average score grouped by metric

It is interesting to note that the combination of Arctic and Mistral models has more of highest scores per question than the combination of Nomic and LLama3 models, but Nomic and LLama3 have better average scores. It should be emphasized that most of the reference texts used in evaluation are references from knowledge. When Arctic obtains good references, Mistral often returns the reference text without changes, but when it doesn’t obtain good references, the evaluation score is low. Even though Nomic obtains better results, LLama3 modifies the result so even if it’s a good result, it doesn’t fully match the reference text used on evaluation, but still the evaluation scores are higher than the ones Mistral gets with bad references.

Conclusion

AI technology is gaining more and more interest from the public and more and more companies want to include it in their businesses. The most common and the most convenient way for them to do it is to get a chatbot, and that is why in this paper there was demonstrated how to develop a chatbot.

The essential terms and concepts, like Vectors, Embedding, Grounding, LLM and RAG, were explained, as well as evaluation metrics.

This application is a simple Blazor web application. Using the application, the user can add the new knowledge to the chatbot, ask him questions, and look at the sources he used to generate his answer. To run the model locally, the “LLamaSharp” library was used. The software development kit “Semantic Kernel” was used for communication with the model and to provide “ChatHistory” class which makes it possible for chatbot to understand context. For calculating embeddings and storing embedding vectors into the data store, the “KernelMemory” service was used.

AI functionalities in this application have been achieved using Large Language Models downloaded from Hugging Face, specifically text embedding models “Nomic” and “Arctic”, and text generation models “LLama3” and “Mistral”. When adding new knowledge to the chatbot, text embedding model is used to calculate embeddings. When asking the chatbot a question, the text generation model is used to generate an answer depending on the prompt.

The chatbot is developed following the technique of Retrieval-Augmented Generation and therefore is limited to answering only questions which can be answered using only the knowledge stored in the data store. If the chatbot is asked other questions, he will give predefined answer that further ensures he doesn't say something he shouldn't.

Although all stated requirements are implemented, there is always room for improvement. For instance, calling APIs could be built in. That would improve the accuracy of the chatbot because he could access real time information. For example, this chatbot which is used as an assistant for a shipping company could access their sailing schedules and return the up-to-date departure time and the price. Next, the chatbot could be upgraded so it that it can recommend tourist attractions for the place tourists are sailing to. Furthermore, a chatbot could be implemented for internal use where employees could use it for searching company documents. That way an employee could say “give me a document which contains …” and the chatbot would tell him which document contains what he is searching for and could give him references to those documents.

This chatbot is developed to answer questions about a shipping company and its activities only, but it can be easily repurposed for any other field a company is invested in.

Bibliography

[1] S, O. M., P, J. J., A, J. O., R, K. T. *AI Based Chatbot for Educational Institutions*. 2024 Ninth International Conference on Science Technology Engineering and Mathematics (ICONSTEM), Chennai, India, 2024, pp. 1-7

[2] Deshpande, A., Dubey, A., Dhavale, A., Navatre, A., Gurav, U., Chanchal, A. K. *Implementation of an NLP-Driven Chatbot and ML Algorithms for Career Counseling.* 2024 International Conference on Inventive Computation Technologies (ICICT), Lalitpur, Nepal, 2024, pp. 853-859

[3] Lescano, N. L., Yamao, E., Sánchez, E. X. V., Santillan, M. A. P. E. *Iterative design and implementation of a chatbot for sexual and reproductive health counseling in Peru*. 2022 IEEE XXIX International Conference on Electronics, Electrical Engineering and Computing (INTERCON), Lima, Peru, 2022, pp. 1-4

[4] Oracle. (2020, October). *What is a chatbot?.* Oracle. https://www.oracle.com/chatbots/what-is-a-chatbot/

[5] Schwaber-Cohen, R. (2023, June). *Vector Embeddings for Developers: The Basics*. Pinecone. https://www.pinecone.io/learn/vector-embeddings-for-developers/

[6] Descartes, N. (2024, February). *Understanding Vector Databases*. CodeProject. https://www.codeproject.com/Articles/5377237/Understanding-vector-databases

[7] Berger, E. (2023, June). *Grounding LLMs*. Microsoft. https://techcommunity.microsoft.com/t5/fasttrack-for-azure/grounding-llms/ba-p/3843857

[8] Keng, B. (2023, September). *LLM Fun: Building a Q&A Bot of Myself*. Bounded Rationality. https://bjlkeng.io/posts/building-a-qa-bot-of-me-with-openai-and-cloudflare/#large-language-models

[9] Luo, Q., Zeng, W., Chen, M, Peng, G., Yuan, X., Yin, Q. *Self-Attention and Transformers: Driving the Evolution of Large Language Models*, 2023 IEEE 6th International Conference on Electronic Information and Communication Technology (ICEICT), Qingdao, China, 2023, pp. 401-405

[10] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., Polosukhin, I. *Attention is all you need.* Advances in neural information processing systems, 2017, pp. 5998-6008

[11] Dey, S. (2024, April). Deep Dive into Self-Attention by Hand. Medium. https://towardsdatascience.com/deep-dive-into-self-attention-by-hand-%EF%B8%8E-f02876e49857

[12] Microsoft. (2024, March). *Blazor | Build client web apps with C# | .NET*. Microsoft. https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor

[13] Hugging Face. (2024, May). *Hugging Face Hub documentation*. Hugging Face. https://huggingface.co/docs/hub/index

[14] SecondState. (2024, April). *Nomic-embed-text-v1.5-Embedding-GGUF*. Hugging Face. https://huggingface.co/second-state/Nomic-embed-text-v1.5-Embedding-GGUF

[15] ChristianAzinn. (2024, April). *snowflake-arctic-embed-m-long-gguf*. Hugging Face. https://huggingface.co/ChristianAzinn/snowflake-arctic-embed-m-long-GGUF

[16] lmstudio-community. (2024, May). *Meta-Llama-3-8B-Instruct-GGUF*. Hugging Face. https://huggingface.co/lmstudio-community/Meta-Llama-3-8B-Instruct-GGUF

[17] TheBloke. (2024, January). *Mistral-7B-OpenOrca-oasst_top1_2023-08-25-v2-GGUF*. Hugging Face. https://huggingface.co/TheBloke/Mistral-7B-OpenOrca-oasst_top1_2023-08-25-v2-GGUF

[18] Microsoft. (2024, May). *semantic-kernel.* GitHub. https://github.com/microsoft/semantic-kernel

[19] Microsoft. (2024, May). *kernel-memory.* GitHub. https://github.com/microsoft/kernel-memory

[20] SciSharp. (2024, May). *LLamaSharp*. GitHub. https://github.com/SciSharp/LLamaSharp

[21] Mamdouh, M. (2023, October). *Mastering ROUGE Matrix: Your Guide to Large Language Model Evaluation for Summarization with Examples*. Dev. https://dev.to/aws-builders/mastering-rouge-matrix-your-guide-to-large-language-model-evaluation-for-summarization-with-examples-jjg

[22] Amanat, M. U. (2024, January). *LLM evaluation with Rouge*. Medium. https://medium.com/@MUmarAmanat/llm-evaluation-with-rouge-0ebf6cf2aed4

[23] Madiraju, P. (2022, November). *Evaluation Metrics in Natural Language Processing — BLEU*. Medium. https://medium.com/@priyankads/evaluation-metrics-in-natural-language-processing-bleu-dc3cfa8faaa5

Abstract

Implementation of an artificial intelligence-based chatbot for the shipping industry

This paper was made to explain the essential terms and concepts needed to understand how a simple chatbot works, to demonstrate its development, and to compare performance of few selected Large Language Models.

The chatbot was developed following the technique of Retrieval-Augmented Generation. The Large Language Models used are downloaded from Hugging Face. The models used for calculating text embeddings are “nomic-embed-text-v1.5-Q5_K_M” and “snowflake-arctic-embed-m-long--Q5_K_M”. The models used for text generation are “Meta-Llama-3-8B-Instruct-Q4_K_M” and “mistral-7b-openorca-oasst_top1_2023-08-25-v2.Q4_K_M”. This application consists of a page for inserting new knowledge and a page for calling a chatbot which also contains a container for used sources.

Keywords: Chatbot; RAG; LLM; vectors; Semantic Kernel; Kernel Memory, LLamaSharp

Sažetak

Implementacija jezičnog asistenta temeljenog na umjetnoj inteligenciji za brodarsku industriju

Ovaj diplomski rad izrađen je kako bi se objasnili osnovni pojmovi i koncepti potrebni za razumijevanje kako radi jednostavan chatbot, da bi se demonstrirao njegov razvoj te da bi se usporedile performanse nekoliko odabranih velikih jezičnih modela (Large Language Models).

Chatbot je razvijen slijedeći tehniku generiranja proširenog dohvaćanjem (Retrieval-Augmented Generation). Korišteni veliki jezični modeli su preuzeti sa Hugging Face-a. Modeli korišteni za računanje smještenja teksta (text embedding) su “nomic-embed-text-v1.5-Q5_K_M” i “snowflake-arctic-embed-m-long--Q5_K_M”. Modeli korišteni za generiranje teksta su “Meta-Llama-3-8B-Instruct-Q4_K_M” i “mistral-7b-openorca-oasst_top1_2023-08-25-v2.Q4_K_M”. Ova aplikacija sastoji se od stranice za unos novog znanja i stranice za pozivanje chatbot-a koja također sadrži kontejner za korištene izvore.

Ključne riječi: Chatbot; RAG; LLM; vektori; Semantic Kernel; Kernel Memory, LLamaSharp