



**Università
di Genova**

**DIBRIS DIPARTIMENTO
DI INFORMATICA, BIOINGEGNERIA,
ROBOTICA E INGEGNERIA DEI SISTEMI**

Navigating the Seas of AI: Deploying Small Language Models on Compact Edge computers for Maritime Applications

Nicolò Guainazzo

Master Thesis

Università di Genova, DIBRIS Via Opera Pia, 13 16145 Genova, Italy
<https://www.dibris.unige.it/>



**Università
di Genova**

MSc Computer Science
Data Science and Engineering Curriculum

Navigating the Seas of AI: Deploying Small Language Models on Compact Edge computers for Maritime Applications

Nicolò Guainazzo

Advisor: Daniele D'Agostino, Davide Ancona Examiner: Giorgio
Delzanno

December, 2024

Abstract

This thesis explores the feasibility of employing Small Language Models (SLMs) in environments with limited or no Internet connectivity, leveraging edge devices powered by batteries. While Large Language Models (LLMs) are more widely recognized, SLMs offer significant advantages in such scenarios due to their lower resource requirements. The use case of this study is maritime navigation, where sailors can use our tool based on SLMs as a digital aid to access sailing directions and plan their routes. In this environment, response immediacy is not critical, as users have sufficient time to query information while navigating and planning activities, making edge devices ideal for running these models. To enhance the quality of responses from SLMs, particularly given their constrained knowledge in maritime contexts, this study adopts the Retrieval-Augmented Generation (R.A.G.) methodology, integrating external information from sailing directions. A comparative analysis is presented to evaluate the performance of various SLMs, focusing on response quality, then the effectiveness of the R.A.G. component, and inference times particularly on edge device with only a CPU.

Additionally, thanks to the parallel research of BSc Stefania Zanetta, who tested the same models on GPU-based systems, this thesis incorporates insights into performance variations across different hardware setups.

Table of Contents

Chapter 1	Introduction	7
Chapter 2	Language models	9
2.1	Introduction to NLP	9
2.2	Large language models	11
2.3	Small language models	13
2.4	Instruction tuning	14
2.5	Model compression	15
2.6	Chat-bot	16
2.7	Retrieval-Augmentation Generation	17
2.7.1	Vector database	19
2.7.2	How the R.A.G works ?	20
Chapter 3	State of the art	25
3.1	Edge hardware	25
3.1.1	NVIDIA	26
3.1.2	Intel	26
3.1.3	Raspberry	27
3.2	Models	27
3.2.1	Phi families	27
3.2.2	Gemma	28

3.2.3	Qwen families	29
3.2.4	Italian SLM	29
3.3	Tool overviews	30
3.3.1	OpenVINO	30
3.3.2	LangChain	31
3.3.3	Hugging Face	31
Chapter 4	MarineChat	33
4.1	SLMs initialization	34
4.2	Dataset	38
4.3	R.A.G	40
4.3.1	Vector index	40
4.3.2	Embedding model	42
4.3.3	Re-ranker model	43
4.4	MarineChat... Assemble!	45
4.4.1	Explanation of <code>create_rag_chain</code>	46
4.4.2	Validation of Parameters	46
4.4.3	Configuring the Retriever	46
4.4.4	Integrating the Re-Ranker	47
4.4.5	Constructing the Prompt	47
4.4.6	Combining Documents and Query	49
4.4.7	Returning the R.A.G. Chain	49
4.5	Implementation of the Chat-bot	49
Chapter 5	Experimental results	52
5.1	Hardware configuration	53
5.1.1	CPU	53
5.1.2	GPU	53

5.2	Evaluation set	55
5.3	Model tested	56
5.4	Evaluation Metrics	58
5.5	Test on CPU	61
5.5.1	Overall quality evaluation	63
5.5.2	Gemma 2 2b-it	66
5.5.3	Phi3.5 mini INT4	68
5.5.4	Phi 3.5 mini INT8	70
5.5.5	Qwen2.5 0.5b	73
5.5.6	Qwen2.5 1.5b	74
5.5.7	Qwen2.5 3b INT8	76
5.5.8	Inference time analysis	78
5.6	Test on GPU	81
5.6.1	Inference time analysis	83
5.7	Discussion	84
	Chapter 6 Conclusion and future directions	86
	Appendix A Dataset	89
	Appendix B Document Excerpt	92
	Appendix C Prompt Template	96
	Appendix D Evaluation set	98
	Bibliography	99

Chapter 1

Introduction

Over the past year, the development and use of small language models (SLMs) has grown significantly and has become one of the most important and fascinating topics in the language model landscape¹ [VNSA⁺24]. This growth of SLMs is due to the search for new methods to solve the problems of the better known large language models (LLMs). The training and execution of LLMs are resource-intensive, both in terms of computation and data. This means that they often need specific and centralized hardware in large quantities. Many steps have been taken to create better and better SLMs and to explore new horizons [LZI⁺24], [LFL⁺24]. An Italian SLMs have not yet been developed from scratch, so in order to achieve the best performance we opted for models trained mainly for the English language, so we have had to rely on English data, and the same goes for interfaces.

The goal of this thesis is to evaluate the performance of SLMs in the context of no or poor Internet connection using an edge device powered by battery, thus justifying their use at the expense of the LLMs [MMN⁺24].

We decided to explore as a use case the idea of posing as a sailor on a medium/small yacht who, before entering an unfamiliar port, has to look up and read the information about it in a pilot book. Looking for information such as typical wind directions, tides, currents, sea depths, dangers in entering the harbor or special rules. We would like to try to replace these actions with interactions with SLM on edge devices, e.g. via an app, or better still with integration into on-board systems such as chart-plotters, perhaps being able to avoid writing but speaking directly to the SLMs. The responses to questions do not need to be immediate in this type of environment, as a user has plenty of time to request information while navigating and planning their actions. This aligns perfectly with the use of edge devices to run these models.

To achieve better results in terms of the quality of responses from the SLMs, aware of their

¹<https://www.forbes.com/sites/lanceeliot/2024/11/08/small-language-models-slm-gaining-popularity-while-large-language-models-llm-still-going-strong-and-reaching-for-the-stars/>

limited knowledge, specifically in the maritime domain, we decided to use the Retrieval-Augmented Generation (R.A.G) a method to incorporate information from an external knowledge, in our case a sailing directions.

Then, once a prototype of such a tool has been developed, we will present a comparative analysis of the results, using a specific framework, for the responses of different SLMs, the R.A.G. component, and the inference times of the SLMs, particularly on an edge device with a CPU.

I am in debt with BSc Stefania Zanetta, my course-mate, who conducted her research in parallel to ours on a similar problem in the context of first aid. We collaborated on the initial selection of SLMs to test, ran preliminary experiments on the same edge device, and each conducted tests on the other's edge devices. This collaboration allowed us to evaluate our tool with various SLMs on an edge device equipped with a GPU.

The structure of this thesis is divided into six chapters, each of which plays a specific role in the development of the work and the analysis of the results obtained. The following is a preview of what will be shown in the body of the thesis.

The second chapter provides a theoretical overview introducing the fundamental concepts of LLMs and SLMs. Two central techniques for project development are also presented: instruction tuning, which improves the capabilities of language models in performing specific tasks, and model compression, a strategy for reducing the computational resources required by models while maintaining their effectiveness. In the last part of this chapter we focus, from the theoretical side, on the key elements underlying our tool: the chat-bot, the program for interacting with a user, and the R.A.G. module, designed to integrate additional information from external sources in order to improve the model's responses.

The third chapter analyses the state of the art for three fundamental areas: the edge hardware that can be used to run SLMs, the models themselves (SLMs) and the tools available for their implementation and optimization.

Chapter 4 goes into details of the implementation aspects of our system, called MarineChat. This chapter is divided into three main sections: the first describes how to use the SLMs, the second illustrates the R.A.G. pipeline, explaining how the integration of external information is managed, and the third shows how the implementation of the union of the two previous parts was done.

In Chapter 5, the results of the experiments performed are presented and discussed. This section describes how the results were obtained, the tools used during the process and the metrics chosen to evaluate the performance of the system.

Finally, Chapter 6 brings together the main conclusions of the work, highlighting the results achieved and the practical implications of the project. This chapter also includes a reflection on possible future extensions, suggesting directions for improvements.

Chapter 2

Language models

2.1 Introduction to NLP

The field of Natural Language Processing (NLP) has developed over nearly seven decades, with its foundations laid in 1950 by Alan Turing, who introduced the Turing Test [Tur09]¹. This test assesses the ability of a machine to produce responses that are indistinguishable from those of a human. If a machine passes, it demonstrates human-like intelligence, sparking early debates in artificial intelligence and motivating research to create machines capable of passing such a test.

NLP's journey has been marked by several cycles of enthusiasm and stagnation. In 1954, Georgetown University and IBM developed a system translating over 60 Russian sentences into English [Hut04], leading researchers to believe that full machine translation could be achieved within a few years.

However, progress slackened, and in 1966, the Automatic Language Processing Advisory Committee (ALPAC) reported that machine translation was costly, inaccurate, and slower than human translation [Hut07], casting doubt on its near-term viability². Funding cuts followed, leading to a near halt in research for almost a decade. Despite setbacks, NLP research resurfaced in the 1970s, driven by increased computational power and lower costs, which made the field more accessible to a wider array of researchers globally.

The late 1980s brought new advancements as IBM's Thomas J. Watson Research Center released the first statistical machine translation systems [BCDP⁺02]. Previously, machine translation relied on labor-intensive rules-based approaches, where handcrafted rules controlled translation accuracy but struggled with edge cases. Statistical machine translation

¹<https://plato.stanford.edu/entries/turing-test/>

²https://nap.nationalacademies.org/resource/alpac_lm/ARC000005.pdf

reduced dependency on these rules, instead using bilingual corpora to learn translations. This data-driven approach gained popularity until neural machine translation emerged in the mid-2010s.

The 1990s saw expansion beyond text with a focus on speech recognition. Early work by Bell Labs and IBM paved the way, but these early systems were limited. By the mid-1980s, IBM’s statistical methods had led to a voice-activated typewriter with a 20,000-word vocabulary, marking a milestone in continuous speech recognition. The late 1980s brought similar successes from DARPA³, Bell Labs, and Carnegie Mellon, with improved vocabularies that rivaled human language capability. In the 1990s, as researchers moved into industry, commercial applications [Hut07] of speech recognition flourished, and companies like Google began hiring speech recognition experts by 2007. Around this time, government agencies, including the NSA, started leveraging speech recognition for tagging keywords in recorded conversations.

By the early 2010s, researchers began exploring deep neural networks for NLP tasks, with early successes seen in long short-term memory (LSTM) networks [VHMN20]. Google used LSTM models in 2015 to enhance Google Voice, leading to significant improvements in NLP applications and drawing more investment into the field. These advances integrated NLP technologies more deeply into daily life, with, for instance, early 2010s car voice recognition evolving from handling limited commands to comprehending broader, more contextual language.

Reflecting on NLP’s progress, the field evolved from rules-based systems to statistical machine translation and, eventually, to neural networks. While research was long an academic pursuit, NLP has recently become widely mainstream. This steady progress has made NLP a central focus within AI, with rapid advancements continuing to redefine the field and make it one the hottest topics in AI.

The real breakthrough moment for the field of NLP came in 2017 with the presentation of the Transformer architecture [Vas17] by Google researchers; the first notable work exploiting this innovation was BERT [DCLT19], also by Google. In 2019, pretrained language models such as OpenAI’s GPT-2 [RWC⁺19] made a splash, generating new content on the fly based on previous content and generalize unseen new task, a previously insurmountable feat. In 2020, OpenAI released an even bigger and more impressive version, GPT-3 [Bro20], building on its previous successes.

³The Research and Development Agency of the U.S. Department of Defence

2.2 Large language models

Large Language Models, or in short LLMs, represent, as pointed out in the previous Section, a significant advancement in natural language processing. They are designed to understand, generate, and manipulate human-like text. To understand how LLMs and also the SLMs work, it is essential to grasp some key concepts: tokens, parameters, prompts, and the general structure of these models [MMN⁺24]. In this section we will only look at these key concepts.

Tokens are the fundamental building blocks used by LLMs. Instead of processing entire words or sentences, LLMs break down text into smaller units called tokens. These tokens can be entire words, parts of words, or even individual characters, depending on the language and the tokenization method used to process the training data [Vas17], [Sen15]. By splitting text into tokens, LLMs can better understand and generate complex language patterns, even handling unfamiliar words by breaking them into recognizable segments. The number of tokens is crucial, as it dictates how much information the model can process in a single run, often with certain limitations.

Parameters are the elements that define the internal structure of LLMs. They are the numerical values that the model adjusts during its training phase to capture linguistic patterns. When referring to parameters, we typically look at the weights and biases of the normal neural networks, which are adjusted during training to minimize the loss function. Parameters determine how the model transforms input data into meaningful output. Essentially, they hold the knowledge the model has gained from vast amounts of training data, allowing it to predict and generate human-like responses. The larger the number of parameters, the more data that make up the training set, and the more computing power, the more capable the model is of understanding nuances and relationships within the language, and thus generally improve its performance [KMH⁺20]; making models like GPT-4o, with estimated 1.7 trillion of parameters ⁴, highly sophisticated.

Prompts are the inputs provided by users to guide the LLM in generating a response. A prompt is the way to communicate with an LLM: it can be a question, a statement, or any text that sets the context for the model to respond to. So we can say that prompting is the practice of representing a task as a natural language utterance in order to query a language model for a response. The quality of the generated response often depends heavily on how the prompt is framed. Crafting an effective prompt is an art in itself, as it can significantly influence the output's relevance and clarity [LYF⁺23], [BSY⁺22]. This is why prompt engineering is emerging as a new focus in the field of NLP. Finally, it is worth noting that each language model usually requires a specific template to writing prompts ⁵. In order to improve the ability to follow prompt instructions, instruction-tuned models

⁴This is only an estimate because OpenAI has never disclosed the true number of parameters

⁵<https://huggingface.co/docs/transformers/en/tasks/prompting>

have been developed, which are discussed in more detail in Section 2.4.

The general working of an LLM involves processing the input prompt, breaking it down into tokens, and then using its parameters to generate a coherent response. When given a prompt, the model assigns probabilities to possible next tokens based on learned patterns, ultimately constructing a complete and contextually appropriate response. This process happens through multiple layers of computations, each refining the model’s understanding and ensuring that the output resembles natural human language.

During this process, the model not only considers the immediate context provided by the prompt but also utilizes its vast network of parameters to draw from related knowledge it has learned during training. Each layer in the LLM architecture builds upon the previous one, gradually improving the representation of the input text and enhancing the quality of the generated response.

One of the key components underlying this architecture is the **transformer** architecture, which relies heavily on a mechanism known as self-attention [Vas17]. The transformer architecture enables the model to efficiently process and generate text by allowing it to focus on different parts of the input sequence simultaneously, rather than sequentially. The self-attention mechanism allows the model to determine which parts of the input are most relevant to each token, effectively capturing long-range dependencies and contextual relationships within the text [XLM⁺23]. This means that the model can understand the importance of each word relative to the others, even if they are far apart in the sentence.

This multi-layered approach allows the model to understand subtleties in language, such as idiomatic expressions, cultural references, and nuanced meanings. Ultimately, the model iteratively refines its predictions, ensuring that the final output is coherent, contextually accurate, and closely aligned with human language patterns.

The most widely used LLM architectures are encoder-only, decoder-only, and encoder-decoder. Most of them are based on Transformer (as the building block). Encoder-only models are designed to deeply understand the input text by processing it bidirectionally. They are called encoder-only because they use only the encoder part of the Transformer architecture to generate a representation of the input text. These models excel at tasks that require a comprehensive understanding of the input, such as classification, named entity recognition, and question answering, where capturing the context of the entire text is crucial.

Decoder-only models are typically used for generating text. They are called decoder-only because they utilize only the decoder part of the Transformer architecture. These models process input sequentially, generating one token at a time while conditioning on previously generated tokens. This makes them ideal for tasks such as text generation, dialogue systems, and completion tasks.

Encoder-decoder models combine both an encoder and a decoder, which is why they are

referred to as encoder-decoder models. The encoder transforms the input into an intermediate representation, and the decoder then uses this representation to generate the output. This architecture is particularly effective for tasks where the output is highly dependent on the input, such as translation, summarization, and other sequence-to-sequence tasks. The vast majority of the most modern and widely used LLMs are decoder-only transformers such as: GPT [Bro20], Llama [Gen23], Falcon [AAA⁺23] models. There are also example of Italian versions of LLMs, such as LLaMAntino [BMP⁺23] and Minerva⁶. However, you may also come across encoder-decoder transformer LLMs, for example Flan-T5 [CHL⁺24] and BART [Lew19].

Encoder-decoder type models, one of the best known is the BERT [Dev18], are typically used in generative tasks where the output is highly dependent on the input, for example in translation and summarization. The decoder-only models are used for all other types of generative tasks.

2.3 Small language models

Small language models [LLC⁺24], [VNSA⁺24] or small large language models⁷, in short SLMs, are reduced versions of the more common LLMs that have gained so much visibility in recent years. Reduced versions first of all in terms of the amount of parameters they contain. Normally they have been integrated into chat-bots (we will see what a chat-bot is in Section 2.6), obviously for their language capabilities that tend to imitate human behavior seemed the most logical application.

SLMs were conceived as a counterpart of LLMs, to overcome what was stated in the so-called *scaling laws* [KMH⁺20]. These scaling laws suggest that the true indication of the model’s capabilities relies on the size of the datasets on which it is trained, computational power, and the number of parameters, given a fixed data source. Through the use of different data sources and higher data quality [GZA⁺23], along with other methodologies such as *knowledge distillation* [TRP⁺24] or new architectures [LZI⁺24] SLMs offer performance comparable to much larger models [ZLCL23], while remaining lightweight enough to be used locally on small edge devices, even without an Internet connection. This feature solves a serious problem in the use of LLMs, related to user privacy. In fact, given the size of the most popular LLMs like ChatGPT, they are always only usable through online service or APIs. Which means that user queries, particularly sensitive ones involving personal or corporate information, could be vulnerable to interception by malicious actors. Another issue with using non-locally based LLMs can be that our queries can be harvested as data

⁶The technical report of this model family is not yet available.

⁷this designation emphasizes the fact that while smaller than common LLMs they remain capable of a “large” understanding of language models. Throughout the text we will use the simpler designation Small language models (SLMs)

to train new LLMs. For example, OpenAI saves chats with its models for 30 days to have new data to improve them⁸. The last case we report, and there are of course many others [Seb23], [YLX⁺24], is when the same data used to improve third-party models or to train new ones is exposed again by other malicious agents through a so-called data leakage attack [LGF⁺23]. All of the above problems are eliminated by using SLM locally. However in this thesis we are more interested in performance and quality with respect to privacy.

Small language models to date are one of the most thriving and fascinating fields of research when it comes to language models, and their capabilities are yet to be fully explored.

2.4 Instruction tuning

A key capability that has made language models important is that you can use them for various tasks by giving them none or a few examples of what you want them to do, [Bro20], [RWC⁺19], without special training. This is why language models have been called *Few-shot learners* or *one-shot learners* or even *zero-shot learners*, depending on the amount of examples you pass through the prompt to the model to explain what you want it to do.

These capabilities sometimes expire in unintended responses, maybe incorrect or misleading: such as inventing facts given as real or generating malicious, toxic text or again and more importantly for our work, models simply do not follow the given instructions via prompt.

This is because models have as their goal to predict subsequent tokens that can produce meaningful text but this collides with the idea that a user has which is that the model must precisely help him in the task he has asked it to do by following its instructions. When a model does not follow instructions it is said to be *non-aligned* with the desires of the human.

An unintended response generated by a language model, often consisting of false or misleading information presented as if it were factual is called, in the field of language models “*hallucination*”, after the phenomenon observed in human psychology. Hallucinations occur when the model generates outputs that deviate from the expected or desired task, leading to unreliable or non-factual results (See also Section 2.7.2).

To solve this problem, the idea of still training the model using a method called *Supervised Fine-Tuning* (SFT) [ZDL⁺23] on datasets called Instruction output (a meticulously annotated set of instructions and response pairs) to enhance the capabilities and improve the controllability of LLMs [CHL⁺24], this technique is also called *instruction-tuning* (IT) [ZDL⁺23], [PLH⁺23] and now is a fundamental part of the normal training stages of any LLMs.

For this work, it is very important to have instructional models, because it is essential that

⁸<https://openai.com/consumer-privacy/>

the models follow the developer’s instructions to accept the context created by the R.A.G. as an addition to his knowledge acquired during training and thus be able to respond better to the user.

2.5 Model compression

To meet the challenge of having language models that can be used not only by accessing services hosted in large data centers over the Internet, but also on edge devices, it has been studied the possibility of compressing the models by reducing the number of their considered parameters. In this way, inference times⁹ of models already running on a hardware, whatever it is, can be improved.

By **compression** we mean something like a loss process. This means that by compressing a model, we reduce its weight in terms of bits, sacrificing some information. Compressed models do not perform as well as their uncompressed counterparts because they lose accuracy in favor of lighter weight, which allows for faster inference. This is similar to an information-loss transformation, where some fidelity is traded off for a reduction in size. In practice, the type of compression we have described is not like a lossless transformation, such as that used to generate a `.zip` file, in which a re-dimensioning of the file weight is not associated with the loss of information.

Various methods for compressing a language model have been proposed and studied in recent years. We can identify five main groups: quantization, pruning, knowledge distillation, compact architecture design, and dynamic networks [WCL⁺24]. Only the first method, quantization, will be used directly in this thesis. The term “*quantization*” refers to a process of converting high-precision values into lower precision values, thus reducing the number of bits required to represent them.

An extreme version of this approach, which aims to drastically reduce the model’s precision while maintaining most of its capabilities, is to reduce the model’s weights precision to a single bit (1-bit LLM)¹⁰ [WZS⁺24].

We will therefore focus on “*quantization*” methods, given the limited resources available for this thesis, it is natural to turn even more specifically toward that part of quantization algorithms known as Post-training Quantization (PTQ). These methods have known wide fame on Hugging Face¹¹, because they do not require re-training the pre-trained model as is done with Quantization-Aware training (QAT) systems. There are several categories of

⁹Inference time for a LLM refers to the duration it takes for the model to process an input and generate a corresponding output.

¹⁰The first usable version of a inference framework that use this approach is available on the Microsoft’s Github repository

¹¹On the Hugging Face site by searching, for example the keyword “quantized” results in 1,438 models as of 10/24/2024

PTQ algorithms:

1. **Equivalent Transformation:** This category deals with the presence of outliers¹² in the weights and activations of language models, which pose a significant challenge to quantization. Outliers force a large number of normal values to be represented with a limited number of bits, causing quantization errors and accuracy degradation. The equivalent transformation aims to mitigate this problem by changing the weights and/or activations before quantization to minimize the loss of information [GDW⁺24]. Activation-aware Weight Quantization or AWQ for short is one of the best-known algorithms of this type.¹³
2. **Weight Compensation:** This technique aims to compensate for the loss of information during quantization by adjusting the remaining weights. An example is GPTQ [FAHA22], which quantizes one weight at a time and updates the un-quantized weights to compensate the error.
3. **Mixed precision:** This strategy addresses the problem of outliers by representing outliers with higher precision and other values with lower precision. The choice of granularity for mixed precision leads to several categories: Element-wise, Channel-wise, Tensor-wise.[GDW⁺24]

As we have already mentioned, in this work we will also try quantized models, but we will limit ourselves to using only quantized SLMs to try to speed up inference time, leaving out the quantization of LLMs in order to be able to run them on edge or local devices. Obviously, this will result in a degradation in the model's response quality.

2.6 Chat-bot

Now we will see how users can interact with LLMs and SLMs. It is possible via a chat-bot. It is now useful to take a brief look at what a chat-bot is and how it has evolved over the years to the present day. The simplest way to define a chat-bot is to think of it as software that attempts to imitate the way a human being speaks in order to interact with the user naturally. Over time, chat-bots have proven useful in a wide range of use cases, from customer service and technical support to healthcare and education. They enable users to ask questions, receive answers, and accomplish tasks with a level of convenience that mimics direct human interaction.

¹²We mean those values that deviate so much from the expected probability distribution

¹³Searching for the keyword “AWQ” among the models on Hugging Face brings up 3,950: <https://huggingface.co/models?sort=trending&search=AWQ>. Also at the date 10/24/2024

Many ways have been tried over the years to implement chat-bots, reflecting advances in technology and artificial intelligence. One of the earliest chat-bot program was created in 1966 and was called ELIZA [Wei66]. It exploited a rules-based approach in which the answers to user questions were already pre-defined following simple scripts. Despite his simplicity, ELIZA appeared as an intelligent conversationalist and fooled many people into believing that the response of their questions was given by another human. Even today in certain contexts these types of chat-bots can be useful.

At the turn of the 2000s the Artificial Intelligence Markup Language (AIML) [MdAB⁺13] was developed that exploited a knowledge bases, allowing for more flexibility in the way they interacted with users. The first chat-bot created using this new language was called ALICE, which represented a significant evolution, as it could understand more complex user input and respond in a way that seemed more fluid and adaptive [Wal03].

The next major leap in chat-bot development was the adoption of natural language processing (NLP) techniques. With NLP, the chat-bots programs were no longer limited to fixed scripts or pre-programmed responses; instead, they could understand the meaning behind user inputs and generate more contextually appropriate replies. The integration of artificial neural networks allowed chat-bots to process language in a more nuanced manner, which led to significant improvements in how natural and helpful these interactions felt [AM20]. Today, the most advanced chat-bots are powered by LLMs and are called LLM-based AI chat-bots, but SLMs can also be used with similar results. These models allow the chat-bots to appear that really understand the user intent, providing relevant answers, and even handle multi-turn conversation without losing the point [DHQZ24]. Chat-bots have now reached a level where they can participate in complex conversations and appear intelligent. The evolution of chat-bots in the last decades is truly extraordinary and is open to new scenarios such as multi-modality ¹⁴ [FZW⁺24].

2.7 Retrieval-Augmentation Generation

We can now move on to the second, equally important element: the Retrieval-Augmentation Generation system, which we will refer to as R.A.G. This system plays a crucial role, since it is the key to overcoming one of the main limitations of SLMs, namely the lack of specific and up-to-date knowledge about highly specialized domains.

Indeed, although SLMs have demonstrated remarkable capabilities in natural language processing and the generation of contextually consistent responses, they are limited by what they have learned during the training phase. Often, information about the datasets on which the models are trained is not fully disclosed; in fact, researchers and developers typically provide only general details about the corpus of data, without fully specifying the individual sources or domains of knowledge contained therein.

¹⁴Multi-modal i.e. able to accept images, audio or video as input

For this reason, it is safe to assume, also on the basis of empirical observations, that the knowledge base of these models does not thoroughly cover very sectoral areas, such as navigation and seaports, which are the focus of our case study. The models are mainly trained to acquire a broad, generalist knowledge, so that they are adaptable to different types of tasks and contexts. However, when dealing with extremely specific topics, their ability to provide precise and detailed answers may be limited or insufficient.

This is precisely where the R.A.G. system comes in. This approach makes it possible to enrich SLMs with external, up-to-date information by integrating data from external or domain-specific sources into the language generation processes. In practice, this means that we can overcome the inherent limitations of SLMs by providing them with access to data that they would not otherwise have. This process of ‘augmentation’ through information retrieval is particularly useful in contexts where the model’s knowledge base of the model is poor or incomplete, as in the case of our maritime domain chat-bot. Thanks to R.A.G., SLMs can therefore also operate effectively in specialized and sectoral domains, improving the quality of the answer and the relevance of the information provided.

Below in Listings 2.1, you can see, as an example, the answer provided by an SLM¹⁵ run on a local system using ollama¹⁶. The question posed to the model is: ‘*What are the typical winds in Genoa ?*’. This is a specific query requiring some geographical and meteorological knowledge concerning the city of Genoa, an important port in the Mediterranean Sea. However, as it can be seen from the example, the answer generated by the model turns out to be very far from what one might call a good answer or, in other words, from what one would expect in a professional or specialist context such as ours.

```

1 Windows PowerShell
2 Copyright (C) Microsoft Corporation. All rights reserved.
3
4 PS C:\Users\
5 ico> ollama run qwen2.5:0.5b
6 >>> What are the typical winds in Genoa ?
7 As an AI language model, I can provide some general information about
     wind patterns in different places around the world, but it is not
     possible for me to provide specific details like "the typical winds
     in Genoa" without being able to access real-time data or a
     geographical database.
8
9 If you have more context about where and when you are interested in this
     information, I can try to give you a more accurate and up-to-date
     answer based on the current conditions.
10
11 >>> \Send a message (/? for help)

```

Listing 2.1: Incorrect answer by an SLM (Qwen2.5 0.5B) to a question concerning our use

¹⁵In this case the SLM used has been the Qwen2.5 0.5b

¹⁶<https://ollama.com/>

case, showing inability to satisfactorily answer questions on issues probably excluded from the knowledge used to train the model.

In the following sections, we will focus on vector databases, which are the persistent and fundamental basis for any R.A.G. system. Although they are not strictly necessary for the creation of a basic R.A.G., they become crucial when one wishes to implement a system that can refer to a large knowledge base, which must be archived and reused several times. This not only optimizes the performance and scalability of the system, but also ensures fast and structured access to a vast amount of information, which would otherwise be difficult to manage efficiently.

We will then explore the theory behind vector databases, highlighting their strategic role in ensuring that the system can run smoothly even in the presence of large amounts of data. Next, we will focus on the architecture of the R.A.G. system as a whole, breaking down all the subsystems that compose it, and analyzing their essential structure and interactions. In this part of the paper, we will keep the focus mainly on the theoretical framework, analyzing the operating principles and methodologies underlying these technologies.

Later in the work, we will focus on the practical part, i.e. the concrete implementation of all the elements that will be discussed here exclusively from a theoretical point of view.

2.7.1 Vector database

The so-called vector databases, in short VecDBs, are a type of database designed to support operations (primarily search and storage, as we will see) on vector-type data. Thus, it can be said that VecDBs are designed to support unstructured data [HLW23], unlike traditional DBMS.

These vectors are generated by applying different forms of transformations or embedding functions that map the input, so the data, into a vector space where the locality implies the semantics meaning of the input. To do this, several method can be used, based on machine learning , word embeddings, feature extraction algorithms or using deep neural network (for instance BERT-like architecture [Dev18] [XLZ⁺23]) normally called embeddings model¹⁷. All these methods can be applied to raw data of various types such as video, images, audio, and text (as briefly shown in Figure 2.1). It is important to emphasize that since these methods are deterministic, if applied several times to the same data, the same vectors will always be obtained as a result. This will be important later, when we explain how the R.A.G. system works in the dedicated section.

¹⁷This last method is the one used in this work

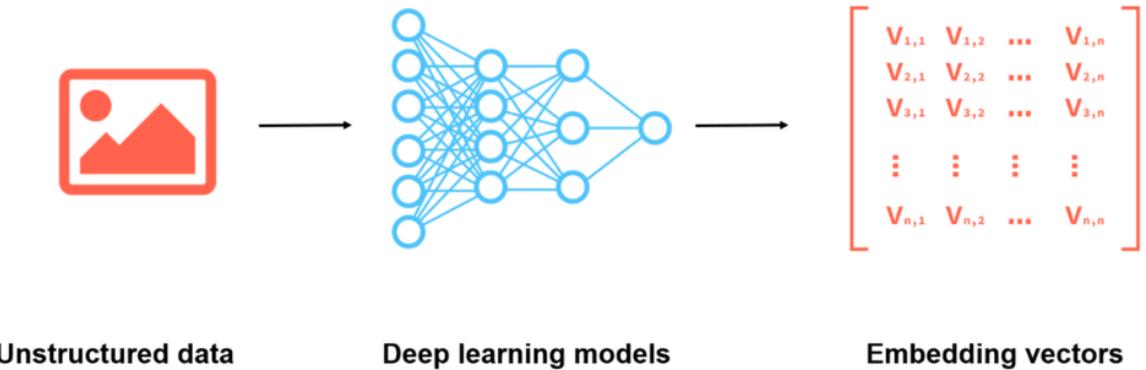


Figure 2.1: A very simple but clear view of how a set of vectors is created to then create a VecDB

These characteristics allow VecDBs to perform searches much faster than traditional databases, even on enormous amounts of data (making them well-suited for large-scale applications), thanks to search algorithms such as Approximate Nearest Neighbour Search (ANNS)¹⁸ or the graph-based Hierarchical Navigable Small Worlds (HNSW) [MY20]. Even the structures used to organize the search, such as KD-trees [Ben75] or ball-trees[Omo89], can enhance performance by improving search efficiency. In fact, these structures are already well-known in traditional databases.

Indeed, there are now many VecDBs on the market, both those created as purely vector-based databases^{19 20 21} and traditional databases that have been updated^{22 23} to support vectors in order to handle unstructured data and thus allow searching on it. This is a clear sign that this technique has gained significant traction in the world of computing, especially in the field of generative AI.

2.7.2 How the R.A.G works ?

As we briefly mentioned before, the primary reason for using a R.A.G. system is to enable models—especially SLMs like ours—to provide useful responses, ensuring they give

¹⁸<https://ann-benchmarks.com/>

¹⁹<https://www.pinecone.io/>

²⁰<https://www.trychroma.com/>

²¹<https://weaviate.io/>

²²<https://postgresml.org/docs/>

²³<https://cassandra.apache.org/doc/latest/cassandra/vector-search/concepts.html>

accurate, factual answers. This prevents them from fabricating unintended responses to questions they are not equipped to answer. Inventing facts given as real or generating malicious or toxic text. This happens because, during the model’s training phase, the training set used did not provide a sufficient knowledge base to address such questions (an example of this can be seen in Listings 2.1). This situation arises frequently when models are queried about knowledge related to niche domains, such as maritime navigation, with its specific rules and challenges.

R.A.G. systems are, therefore, among the various approaches designed to mitigate this phenomenon [TZJ⁺24], [HYM⁺23], commonly referred to as model “*hallucination*” [HYM⁺23], [ZLC⁺23] (See also Section 2.4). There are many categories in which hallucinations can be classified ²⁴, for example two major categories are Factuality Hallucination and Faithfulness Hallucination. The first one occurs when the model generates content that is factually incorrect or deviates from real-world truths. In contrast, faithfulness hallucination arises when the generated output, while potentially accurate, is not aligned with or supported by the input or source content, reflecting a lack of fidelity to the provided material [HYM⁺23]. The R.A.G is a paradigm that helps mitigate the problems of factual hallucination [JLF⁺23].

The R.A.G. not only helps address the problem of hallucinations, but it is also a tool that can be used to integrate more recent knowledge compared to that used to create the dataset during the model’s training phase [MAR24]. This avoids the need to retrain the model, which is always a costly operation, in order to update it with more recent information.

The R.A.G can thus be seen as a non-parametric memory [LPP⁺20] to go along with a Language Model that increases its performance and avoiding having to train or fine-tune it on a specific topic, all time-consuming and expensive operations.

In recent times, many different tools, including open-source ones ²⁵ ²⁶ ²⁷, have been developed to enable the use of R.A.G. systems. Many of these tools are excellent and widely used. However, for this work, it was decided to develop a custom version specifically tailored to the objectives of this project, rather than relying on pre-implemented tools. Therefore, the focus was placed more on the testing phase rather than on user experience.

An R.A.G system comprises two core components: the retriever and the generator [ZZY⁺24], as illustrated in Figure 2.2. The process takes place, in principle, as follows: The retriever (the first core components), after considering the user’s question/query, searches for the most relevant information, which, once found, is passed along with the initial question-/query to the generator part (the second core components), which can then finally answer it. The above description is deliberately , a bird’s eye view of how an R.A.G. system works. In the implementation explanation in Section 4.3 we will go into more details.

²⁴<https://github.com/LuckyyySTA/Awesome-LLM-hallucination>

²⁵<https://anythingllm.com/>

²⁶<https://lmstudio.ai/>

²⁷<https://www.jan.ai/>

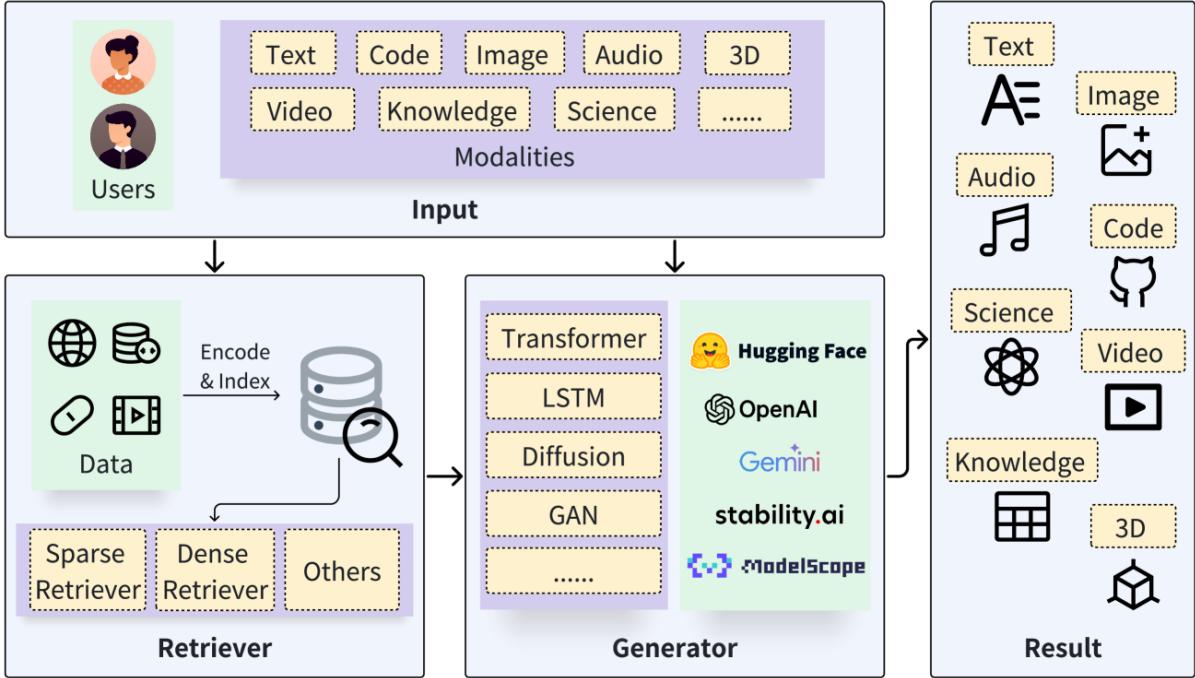


Figure 2.2: Here an example of RAG diagram

Since the goal of this work is to create a chat-bot, the generator will utilize a transformer model, more specifically, as we have already mentioned, an SLM, which specializes in generating only natural language. However, it's important to note that this is not the only possible approach when building a R.A.G. system.

There are at least three other types of models that can serve as generators in a R.A.G. framework. These include Long Short-Term Memory (LSTM) models[VHMN20], which are known for their ability to handle sequential data and maintain memory of past inputs over longer sequences. This makes them suitable for certain types of generative tasks, especially when temporal or sequential relationships are important.

Another option is Diffusion Models[YZS⁺23], which have recently gained popularity for their generative capabilities, especially in image and data synthesis. These models work by iteratively denoising a sample to generate new data, and while they are primarily used in fields like computer vision, they are increasingly being explored for text generation as well [TWZ⁺23].

Lastly, Generative Adversarial Networks (GANs)[GPAM⁺20] represent a third approach. GANs consist of two neural networks—the generator and the discriminator—that work in

tandem. These two parts compete with each other through a process of adversarial learning (derived from game theory), so that the generator can produce more realistic examples, while the discriminator continues to improve its ability to distinguish between true examples and those generated by the generator. Various types of data such as images, audio but also text can be generated via GANs.²⁸.

Each of these models offers unique strengths and potential applications, but for the purposes of this work, the transformer-based SLM was chosen due to its efficiency and effectiveness in generating natural language text, which aligns directly with the chat-bot's requirements.

On the other hand, as far as the retriever part is concerned, i.e. the information retrieval part, usually the first $\text{top-}k$ most relevant information is found according to the similarity function used. There are several types, the main ones being sparse retrieval and dense retrieval, then there are other minor types [ZZY⁺24].

In the first method, *sparse retrieval*, key/value pairs are used to represent the documents to be searched. This method relies on term-matching metrics such as TF-IDF [RW97], query likelihood [LZ01], or BM25 [RZ⁺09]. However, these methods are not used in this work because they rely on exact term, which can lead to retrieval losses due to their inability to recognize synonyms or paraphrased language, and also to explore new methods. Instead, we rely on the second method, called *dense retrieval*, which differs by using embedding systems (BERT models) [KOM⁺20], [MTMR22] to produce vectors from the initial query. As discussed previously regarding VecDBs, vectors are used to create indexes with Approximate Nearest Neighbor (ANN) to speed up the search, which is then performed by calculating the similarity functions to retrieve the top-k most similar vectors between those generated from the documents stored in the VecDBs. This part will be further elaborated in the implementation sections. Finally, besides the first two methods, there are alternative methods for retrieving relevant information [WHW⁺22], [ZWC⁺24]. Like the first, these are mentioned for completeness but are not used in this work.

One final problem remains: how do the retrieval and generation components effectively come together? Or, more specifically, how does the retrieval component augment the generator? There are four different methods to solve this problem: *Query-based R.A.G*, *Latent Representation-based R.A.G*, *Logit-based R.A.G*, and *Speculative R.A.G* [ZZY⁺24]. The method adopted in this work, for its simplicity, is the first one mentioned, which is based on the idea of augmenting the prompt (a kind of prompt engineering) by inserting the most relevant information retrieved directly into the prompt along with the user's original query, before passing it to the generator. This creates a new composite input, which the generator processes while considering the context formed by the information retrieved, which it must base its answer on (see Figure: 2.3). This method is prevalent

²⁸<https://eagleyes.ai/post/the-use-of-generative-adversarial-networks-gans-with-retrieval-augmented-generation-rag-a-comprehensive-exploration>

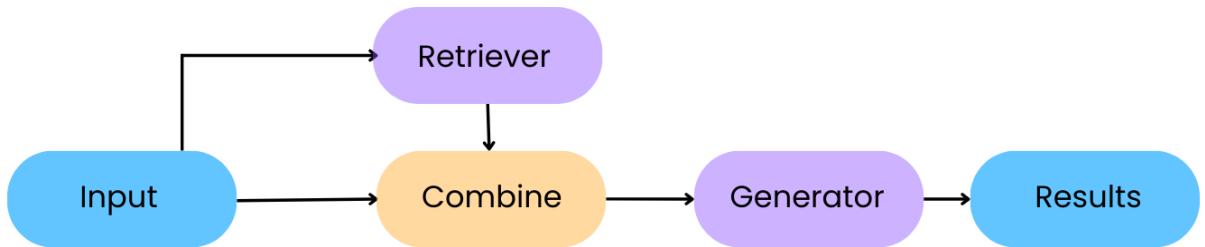


Figure 2.3: Query-based R.A.G diagram

in RAG applications and has shown significant results, particularly in Knowledge Base Question Answering (KBQA) [YYH⁺21].

However, it is necessary to highlight one key issue resulting from the integration of R.A.G. in a chat-bot: the increase in response time due to the addition of numerous intermediate steps, as discussed earlier, before the generator provides the final answer. This issue becomes evident when comparing questions that require a document search using the R.A.G. system to those where the answer does not need to integrate the knowledge base through retrieval ²⁹. In order to be able to avoid having to fine-tune a model for the maritime contest and at the same time avoid, or rather mitigate hallucinations this is the least price one can pay.

²⁹A question like: "What is the capital of Italy?" requires 23.2 seconds, while "What are the typical winds in Genoa?" which, to generate a correct answer, requires finding relevant documents with the retriever, takes 41 seconds. These times refer to the use of Phi-3.5-mini-instruct quantized to Int4 as the generator.

Chapter 3

State of the art

An important element to be introduced in this work is to identify the state of the art elements as they pertain to this project, in order to have a clear idea of the context in which we will move. In this chapter we will show the state of the art in three areas: edge/local hardware, SLMs and tools/frameworks.

3.1 Edge hardware

There is no single, universal definition of what an edge device is. However, it is usually considered as hardware that is close to where data are acquired and does not rely on a centralized system with a very expensive workstation to operate, as opposed to cloud/-centralized architectures. This technology is essential in the context of the Internet of Things and has developed in many different sectors, such as medicine, smart cities or even manufacturing.

What we mean in this thesis is slightly different and should be seen from the perspective of generative AI. The models of the major players in this field inevitably rely on centralized systems. Huge data centers equipped with the latest graphics cards that require a lot of energy, effort to maintain and therefore money. This is the only effective way to train these truly gigantic models and then make them available to a wide audience. However, if we consider only inference on small models (training even SLMs requires significant hardware, time and therefore money), less than 10 billion parameters, it is possible to use edge devices, which we will focus on in Chapter 5. So not simple sensors, which are widely used in classical IoT, but also not powerful clusters. We will now look at three different edge devices from different manufacturers (Nvidia, Intel, Raspberry) that are considered to be state of the art. In this work, we consider two of them: an earlier version of NVIDIA's Jetson family of devices and an Intel Core Ultra processor are used for the reasons just

mentioned.

3.1.1 NVIDIA

The state of the art for NVIDIA is the Jetson module called Jetson AGX Orin 64 GB with a GPU with 2048-core Ampere architecture¹ GPU with 64 tensor cores with a total capacity of 275 TOPS (Tera operations per second) in int8 precision.

The power consumption varies depending on the workload from 15 W to 60 W. The cost of one module is 1999\$ ².

We know how much the emergence and growth of Large language models is also due to the extensive use of GPUs for all the most important steps in the life of an LLM from training to inference, and how much this has benefited the NVIDIA market. So certainly the AGX Orin series board with its GPU has a secure place among the state of the art in this field. For the purposes of this thesis, it can be stated that, at least in terms of inference time, this board performs optimally.

3.1.2 Intel

The second edge computer considered here is Intel's UP Xtreme i14 ³ with Core™ Ultra 7 processor 165H (24M cache, up to 5.00GHz). This module is the state of the art CPU for language model inference and is part of Intel's evidence of maintaining a share of the market in the Generative AI arena for edge/local contexts. The UP Xtreme i14 not only introduces the new Intel® Core™ Ultra processor, which benefits from a new high performance CPU core architecture, but also integrates the new LPDDR5 SDRAM.

The power consumption is typical about 67-80 W. The cost of one of it is 749\$ excluding taxation ⁴.

This combination enables the board to execute heavy task loads and real-time data processing with ease. It provides up to 32 TOPS for advanced industrial robotics, frictionless retail, and smart healthcare. Its AI performance is enhanced by Intel® DL Boost and the Intel® distribution of the OpenVINO™ toolkit, the latter being a key element in the development for the purposes of this work and will see it in more detail later.

¹<https://www.nvidia.com/en-us/data-center/ampere-architecture/>

²<https://www.arrow.com/it-it/products/945-13730-0055-000/nvidia>

³UP Xtreme i14

⁴<https://up-shop.org/up-xtreme-i14-series.html>

3.1.3 Raspberry

The Raspberry Pi 5⁵ features the Broadcom BCM2712 quad-core Arm Cortex A76 processor at 2.4GHz, making it up to three times faster than the previous generation. With up to 8GB of LPDDR4X-4267 SDRAM, this is the fastest and smoothest Raspberry Pi. Raspberry is usually known as the most famous and used module for edge environments, and because it can do inference for SLMs anyway, which is what we are interested in here. Recently, it was added the possibility to integrate the Raspberry Pi 5 with a new board named Raspberry Pi AI HAT+⁶ to use a neural-network accelerator, the edge-AI processor HAILO 8⁷ that increases the computing capabilities to 26 TOPS, so the total expenditure of the two pieces of hardware is 215€ (Raspberry Pi 5 alone costs 90€).

3.2 Models

In this section we described the state of the art of the SLMs with open weights or open source [LLC⁺24], [ZLCL23]. Strictly speaking, the number of truly open source language models is a small part of the entire LM world⁸, but in the Gen-AI community, normally any model that can be used for free is considered ‘open source’. Obviously, this leads to misunderstandings.

We will only focus on models that are directly born as SLMs, and not on larger models that have been reduced in size by some form of compression and are therefore considered “*small*”. This is simply for the sake of brevity, since the number of models of this type is so great⁹¹⁰ and it is often difficult, if not impossible, to distinguish between the best ones and the usefulness of using these models when there are already those designed as SLMs from the outset.

3.2.1 Phi families

This is a family created by Microsoft that started with the phi-1 model and today includes the more powerful phi-3.5 models [AJA⁺24] in its different versions (mini, vision and MoE).

⁵<https://www.raspberrypi.com/products/raspberry-pi-5/>

⁶<https://www.raspberrypi.com/products/ai-hat/>

⁷<https://hailo.ai/products/ai-accelerators/hailo-8-ai-accelerator/#hailo8-overview>

⁸<https://github.com/eugeneyan/open-l1ms>

⁹On the Hugging Face website, for example, if you search for models with 4-int precision, there are more than 17,000 of them.

¹⁰https://huggingface.co/spaces/Intel/low_bit_open_llm_leaderboard

Here, we only considered the phi-3.5 mini instruct model, which is definitely one of the state of the art SLMs because it reaches a value of 69 in Measuring Massive Multitask Language Understanding, a benchmark designed to evaluate the broad knowledge and problem-solving abilities of language models across a wide range of subjects, in short, MMLU (5 shot) [HBB⁺21], which is one of the best-known metrics to evaluate the language models and achieves an MMLU-PRO (0-shot, CoT) [WMZ⁺24] (an evolution of the first metrics) score of 47.5 as well.

More specifically, the phi-3.5-mini model ¹¹ [AJA⁺24], which is the model chosen to be used with regard to this family, is a dense decoder-only transformer and is 3.8 billions parameters in Brain float 16 (BF16) precision ¹² and thus weighs about 7.6 GB. Although small, it is a multilingual model, with good levels in German, French, Italian, etc ¹³. It also supports a context length of up to 128 k tokens, making it suitable for applications where R.A.G is used, as in our case, and demonstrated good capabilities in understanding long text.

It was trained on a 3.4 trillion token dataset already used for phi-3, including data from various sources such as public documents, code, new synthetic data specially created with LLM ¹⁴ and finally human supervised chats. A really big cluster of 512 H100-80G GPUs was used to train the model for 10 days. Finally, the phi-3.5-mini model is used as an instruct model, which allows the model to better adapt and follow human commands in some more specific domains. This feature is again very useful when using RAG methods, as already mentioned in Chapter 2.

3.2.2 Gemma

The second model considered state of the art is Google’s Gemma 2 2b ¹⁵ [TRP⁺24] with, for instance, an MMLU equal to 52.2. It is part of the Gemma family was built from the search also done by Google for Gemini models. This family of language models was designed by Google to allow the use of small language models with limited resources. This is exactly what we are looking for in this project. This once again highlights the growing importance of the “*SLM world*”. The model here considered as SLM is the smallest from 2 billions of parameters with BF16 accuracy. It is a text-to-text language model with a decoder-only architecture, knowing only the English language, with open weights for both pre-trained and instruction-tuned variants. It was trained on a training set of 2 trillion of tokens derived from various sources in particular web documents, code and scientific

¹¹<https://huggingface.co/microsoft/Phi-3.5-mini-instruct>

¹²At thislink the Google’s presentation of the Brain floating point format.

¹³Naturally at a lower level than English. See the Table at this link

¹⁴data updated to October 2023

¹⁵2b stands for 2 billions of parameters.

articles. The model has a context length of 8192, so much smaller than the other models considered in this chapter.

One of the goals of the development team was to avoid training the model on huge amounts of data (finding large amounts of good data is always a problem), and to focus on other methods to improve model performance, such as knowledge distillation[HVD15]. The model has a weight of 4.9 GB, which is slightly less than the phi-3.5-mini model discussed earlier. The hardware used for training was an unspecified number of Tensor Processor Units¹⁶, an AI accelerator developed by Google in 2015.

3.2.3 Qwen families

The third and last family considered here, which makes available models with less than 10b parameters of state-of-the-art, are the Qwen family, specifically the latest version Qwen 2.5 [YYH⁺24], [Tea24] developed by the AI lab of the Chinese company Alibaba Cloud. There are 4 models that can be considered as small language models. Of 7, 3, 1.5 and 0.5 billions of parameters. So in general, smaller models than those seen before. All models were trained on a dataset of 18 billions tokens, other information was not disclosed. Each model knows 29 different languages (including Italian, French, Japanese, Arabic, etc.) and is able to support a context length of 128 thousand tokens, as in the case of phi-3.5-mini. In addition, compared to the previous models of the 2 series, the resistance to various system requests has been improved and, in the case of the new model with 3 billions parameters, the development team claims to have achieved an MMLU of 65.6 for the non-instruction version, while for the instruction version it achieves an MMLU-Pro equal to 34.6¹⁷, The values for the other models are shown in table 3.1. Unfortunately, no information has yet been provided on the hardware used to train all these models.

Datasets	Qwen2.5-0.5B	Qwen2.5-1.5B	Qwen2.5-3B
MMLU	47.5	60.9	65.6
MMLU-pro	15.7	28.5	34.6

Table 3.1: Models performance on selected datasets. <https://qwen2.org/qwen2-5/>

3.2.4 Italian SLM

There is also an Italian SLM's family developed by the Semantic Web and Personalization Research Lab, a laboratory of research within the Department of Computer Science at the

¹⁶Introduction to Cloud TPU

¹⁷Qwen2.5-LLM: Extending the boundary of LLMs

University of Bari, Italy. The first version was called LLaMAntino-2-chat-7b-hf-ITA¹⁸ and is based on the LLama 2 7b model. The second is called LLaMAntino-3-ANITA¹⁹ instead based on the newer llama 3 8b. Both of them are a fine-tuned version of the starting models. So are not totally Italian SLMs [PBS24], [BMP⁺23]. It was then decided not to use them because the dataset with which the R.A.G. system was to be augmented was an English document and, to be consistent, models trained primarily in that language were used. Also, if we had chosen to use Italian SLMs, we would have had the problem of determining whether any problems were due to imperfect language support or the limitations of the SLMs.

3.3 Tool overviews

In a few years, many tools have been born and grown for every field of study and development of applications with Generative AI. By tools we mean mainly libraries or frameworks or sets of them. Many of these have been developed thanks to a community of professionals or simple amateurs, enthusiastic about the results achieved by the branch of so-called non-classical artificial intelligence, on the wave of great popularity gained by the distribution to a wide audience of OpenAI²¹ models. In this section we will see only a small part of these tools (the most popular), though there are now so many tools available²².

The first tool we will look at in this section is OpenVINO, which has been useful in speeding up the time it takes to derive models and transform them into the correct format for use on Intel CPUs. The second tool we will see is LangChain, a framework for building pipelines for projects based on language models. With this tool, we can create the R.A.G. system and integrate it with a chat-bot based on SLMs. The last one is a set of libraries that are useful and integrated with the previous two tools to download models.

3.3.1 OpenVINO

OpenVINO²³ (Open Visual Inference and Neural Network Optimization) is an open-source toolkit developed by Intel. It is designed to optimize and deploy deep learning models on Intel hardware, including CPUs, GPUs, and VPUs. OpenVINO is written in C++ but also offers an API in Python with maximum coverage, whereas C and NodeJs are limited

¹⁸<https://huggingface.co/swap-uniba/LLaMAntino-2-chat-7b-hf-ITA>

¹⁹There are other versions of this.

²⁰<https://huggingface.co/swap-uniba/LLaMAntino-3-ANITA-8B-Inst-DPO-ITA>

²¹ChatGPT 3 was released on 30 November, in five days reach over 1 million users.

²²At this link a list of libraries only for inference of languages models: <https://github.com/vince-lam/awesome-local-llms>

²³OpenVINO's website

only to the most basic methods. The toolkit enables developers to accelerate inference performance, particularly in computer vision, natural language processing, and speech recognition tasks. OpenVINO supports integration with the most important tools for using machine learning models such as ONNX, pytorch, tensorflow and Optimum intel so allowing access to even the latest Language models. It also allows models to be converted after training into a format called Intermediate Representation (IR) to improve efficiency by compressing the weights of the selected model using OpenVINO’s Neural Network Compression Framework (NNCF) [KLS⁺21]. It’s widely used in edge computing scenarios where low latency and efficient performance are critical.

3.3.2 LangChain

LangChain²⁴ is an open source framework for building applications around LLMs and also SLMs. It simplifies the process of integrating LLMs into complex workflows, making it easier to manage different components such as prompt engineering, text splitters, memory management and interaction with external data sources such as vector databases or other third-party libraries (such as OpenVINO), and of course to integrate language models. LangChain supports a variety of use cases, including chatbots, automated content generation and R.A.G.. It is particularly useful for developers and researchers who want to experiment the development of language model-based systems, providing tools that can handle the intricacies of LLM interactions; this is one of the most well-known and most used open source frameworks in the Gen-AI field. ²⁵.

3.3.3 Hugging Face

Hugging Face²⁶ is an AI and NLP (Natural Language Processing) company, known for its extensive collections of libraries and tools for many applications. Hugging Face provides a wide range of pre-trained models ²⁷ to a community of non-professional developers, so they can perform tasks such as text classification, translation, embedding, sentiment analysis, computer vision tasks and more. Its ecosystem includes model hosting, deployment tools and different types of datasets. Hugging Face has become a central platform for NLP and machine learning practitioners, providing easy-to-use APIs and a collaborative environment to advance AI research and applications. Today, HuggingFace is recognized as the most

²⁴LangChain’s website

²⁵Normally its antagonist is LlamaIndex. <https://www.llamaindex.ai/>

²⁶Hugging Face’s website

²⁷<https://arstechnica.com/information-technology/2024/09/ai-hosting-platform-surpasses-1-million-models-for-the-first-time/>

popular²⁸ and used hub for downloading LLMs and SLMs.

²⁸<https://collabnix.com/what-is-hugging-face-and-why-it-is-damn-popular/>

Chapter 4

MarineChat

In the previous chapters we have discussed the components and ingredients of the tool which represents the main outcome of the thesis. In Figure 4.1, we can see the architecture of the R.A.G. chat-bot. We decided to give the tool a name and opted for “**MarineChat**” which highlights two elements of this tool, namely being a chat-bot for the maritime domain. In this chapter, we will focus only on the development of the tool for CPUs (Intel). With regard to the development of devices that can take advantage of the GPU, we refer to the work of my colleague BSc. Stefania Zanetta. The steps we took were:

1. Identified a set of models
2. Created the R.A.G and chat-bot for each other use-case
3. Downloaded the models on CPU and GPU
4. Tested the models on CPU and GPU

For the development of MarineChat, we used three tools: OpenVINO, HuggingFace and Langchain. Each of them was mentioned before in Section 3.3. Then we will see in the next sections the development of the chat-bot part and finally the one of the R.A.G part.

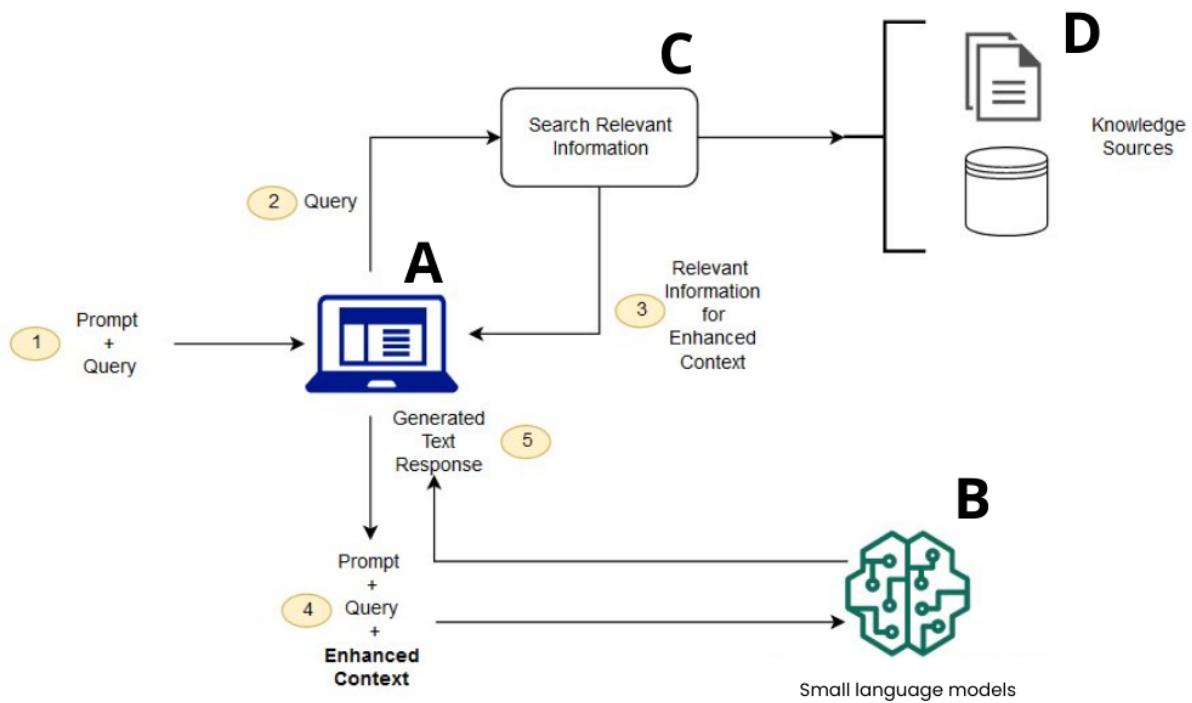


Figure 4.1: General view of MarineChat elements.

4.1 SLMs initialization

The underlying model of the chat-bot is a SLM (we will see later which ones have actually been tested and the results); see in Figure 4.1 the element B. For now, we do not care to identify a specific one, whatever it is, it is used via the class `HuggingFacePipeline`¹ provided by the LangChain² framework, using its integration with the Hugging Face's library transformer, which creates a pipeline. A Hugging Face pipeline is an object that abstracts the most complex steps to run a model, providing a simple way to perform various tasks such as Named Entity Recognition, Masked Language Modelling, Sentiment Analysis, Feature Extraction and, for this work, text generation through a SLM.

To streamline the initialization and configuration of the pipeline, we created a function called `initialize_openvino_pipeline`. This function wraps the setup process for the Hugging Face model pipeline, making it more accessible for repeated use. The function,

¹HugginFacePipeline's API reference

²The LangChain's version used in this thesis is 0.2.16

shown below, takes three parameters: `ov_config`, which is the OpenVINO configuration; `model_id`, an optional model identifier (with a default set to `Qwen2.5-0.5B-Instruct`); and `max_new_tokens`, an optional limit on the number of tokens generated. Here's the code:

```

1 def initialize_openvino_pipeline(ov_config, model_id =
2     "../model/Qwen/Qwen2.5-0.5B-Instruct", max_new_tokens=140):
3     """
4         Initialize the openvino pipeline
5     Args:
6         ov_config: openvino configuration
7         max_new_tokens: maximum new tokens
8     Returns: openvino pipeline
9     """
10    logging.info("Loading openvino pipeline")
11    try:
12        ov_slm = HuggingFacePipeline.from_model_id(
13            model_id=model_id,
14            task="text-generation",
15            backend="openvino",
16            model_kwargs={"device": "CPU", "ov_config": ov_config},
17            pipeline_kwargs={"max_new_tokens": max_new_tokens},
18        )
19        logging.info("Openvino model: {}".format(ov_llm.model_id))
20    except Exception as e:
21        raise ValueError("Error loading openvino
pipeline{}".format(e))
22    return ov_slm

```

Normally the pipeline directly downloads the model with the name passed as a parameter to the function each time the function is called; but if the model is already saved locally, the pipeline also accepts a local model. This is obviously important for the scope of this work. The model needs to be downloaded in the presence of an Internet connection and then saved within the device's file system to be used in the edge environment without a connection. To do this, the Optimum library was used with its integration with Intel, but it will be seen in Section 5.3 with more details, because this section focuses only on the way the model runs and how to create the chat-bot.

Another very fundamental feature of this pipeline is that it allows OpenVINO to be used as a back-end inference engine; via the `ov_config` parameter the engine can be configured. This parameter is a dictionary that contains specific configurations optimizing the pipeline for performance, precision and resource management. Here is a brief description of the key entries within `ov_config`:

- `KV_CACHE_PRECISION`: Sets the precision level of the key-value caching mechanism [PDC⁺23] to `u8` (8-bit unsigned integers), which helps reduce memory usage while

preserving performance.

- **DYNAMIC_QUANTIZATION_GROUP_SIZE**: Specifies the group size for dynamic quantization as 32. This parameter enables quantization that adjusts in real-time, balancing accuracy and efficiency for different processing loads.
- **PERFORMANCE_HINT**: Configured as LATENCY, this option prioritizes low-latency processing, making the pipeline responsive to real-time queries, an essential feature for interactive chat-bot applications.
- **NUM_STREAMS**: Set to 1, this parameter limits the number of concurrent inference streams. A single stream is optimal for CPU resource management and ensures consistent performance across different devices.
- **CACHE_DIR**: This field is set as an empty string here, indicating that no caching directory is specified, then the pipeline use the global cache. A caching directory can be used to store intermediate computations or model data, enabling faster subsequent inferences.

In order to use OpenVINO, it is recommended to convert the model to Intermediate Representation format, in short IR³. OpenVINO does not directly have a tool internally to compress or quantize models but relies on an external package called the Neural Network Compression Framework (NNCF)⁴ [KLS⁺21]. Conversion of the model is recommended because it reduces the size of the model, thus reducing inference time and dependencies. In addition, a model can be quantized to reduce its size and thus improve inference time, obviously losing “accuracy”, and then still be transformed to IR format, this possibility is very important in an edge context. Again, this part about quantizing the model and transforming it into the right format to exploit the use of OpenVINO will be seen in Section 5.3.

So the `initialize_openvino_pipeline` function, once called, returns a `HuggingFacePipeline` object, allowing for easy customization through pipeline arguments. These arguments can be adjusted to fine-tune the behavior of the pipeline to meet specific requirements. In this work, we configure the pipeline with a set of arguments, as shown below:

```
1  pipeline_kwarg = dict(  
2      max_new_tokens=1024,  
3      temperature=0.7,  
4      do_sample=0.7 > 0.0,  
5      top_p=0.9,  
6      top_k=50,  
7      repetition_penalty=1.1,
```

³OpenVINO documentation about IR format

⁴NNCF’s Github repository

```

8     tokenizer=ov_llm.pipeline.tokenizer,
9     skip_special_tokens=True,
10    )
11

```

Here's a brief explanation of each parameter in `pipeline_kwargs`:

- **max_new_tokens**: Sets the maximum number of new tokens generated in response to a user's query. Here, it's set to 1024, providing flexibility for generating long outputs if needed but not too long.
- **temperature**: Controls the randomness of predictions. With a value of 0.7, the pipeline will produce outputs with moderate creativity, balancing deterministic and diverse responses. Normally, it is the best choice for a conversational AI, where the response should be engaging but still meaningful.
- **do_sample**: Determines whether sampling is applied for text generation. Here, `do_sample` is set to `True` (since $0.7 > 0.0$), enabling the model to produce varied responses rather than "safe" ones.
- **top_p**: Implements nucleus sampling, where only the top p probability mass (cumulative probability of 0.9 here) of token options is considered. This parameter was chosen to ensure creativity of response.
- **top_k**: Limits the token selection to the top k choices (50 here). This values was chosen to limit the number of tokens considered by the `top_p`. This parameter refines the model's output by reducing random token generation and enhancing response quality.
- **repetition_penalty**: Penalizes repeated tokens to reduce redundancy. A value of 1.1 discourages the model from repeating phrases or words, ensuring more coherent responses.
- **tokenizer**: Specifies the tokenizer used for pre-processing text. Here, the pipeline reuses the tokenizer from `HuggingFacePipeline` object, ensuring consistency with the model's expected input and output format.
- **skip_special_tokens**: Ensures that special tokens (e.g., `[CLS]`, `[SEP]`) are omitted from the output, making responses cleaner and more readable. This parameter was added to avoid a step in the processing part of the answers in preparation for the tests.

No parameters such as `hide_full_prompt` were used to show only the chat-bot's response without the system prompt for the sake of checking the results and then for future tests, since MarineChat is intended for research purposes.

Once the `HuggingFacePipeline` object has been created, all that remains is to integrate it with the R.A.G. part to finally create the chat-bot R.A.G (See element A in Figure 4.1). We will see how this has been implemented in the next sections.

4.2 Dataset

As been said various time before the R.A.G system needs a dataset also called “retrieval bases” that are the external knowledge added by the R.A.G itself to the model’s knowledge (See in Figure 4.1 the element D). This external knowledge can correspond to different types of resources, such as images, audio or simple text. For this thesis the World Port Index (WPI)⁵ has been used as external knowledge. It is an on-line database of world-wide maritime port information which serves as a sailing direction; a general reference and navigational planning tool for mariners. The WPI provides the general geographic location with over 100 key characteristics and services of thousands of ports around the globe. The principal sources of information in the WPI are the Sailing Directions and charts published by the National Geospatial-Intelligence Agency (NGA), but where information from those sources is lacking or incomplete, other authoritative sources, both domestic and foreign, are used.

The selected portion of the entire WPI focuses on the Western Mediterranean⁶, it is the eighteenth edition, dating back 2022. Specifically, only the Italian coastline has been chosen, corresponding to chapters 10 to 20, from the border with France all the way down the peninsula to the last Italian port in the Adriatic, Trieste, before the national border. This choice was made to start with a smaller (from 508 pages to 269)⁷, more manageable source for testing, with the potential to expand to other locations in the future. For a clearer view of the document, some pages have been reproduced in Appendix B in the form of an image to keep the layout unchanged.

The source provides detailed yet concise information on various aspects of ports as well as the coastline and bays. It is organized by sectors, each representing a maritime area that does not correspond to the political regions of Italy. The document provides various types of information that can be summarized as follows:

- **Geographic and Structural Descriptions:** the document provides comprehensive descriptions of prominent landmarks, natural features, and port structures that are vital for vessel orientation and navigation. These descriptions aid in identifying key

⁵<https://msi.nga.mil/Publications/WPI>

⁶<https://msi.nga.mil/api/publications/download?key=16694491/SFH00000/Pub131bk.pdf&type=view>

⁷The link to access the file on Google’s Drive

points along the coast and serve as references for safe travel and docking (See the aspect paragraph in Figure B.3).

- **Weather Patterns and Sea Conditions:** detailed weather information is included, covering prevalent winds, tidal currents, and seasonal variations in visibility and atmospheric conditions (see Figure B.2). This data is invaluable for mariners and the R.A.G. system as it informs responses to queries on expected conditions, helping users plan safe and efficient routes.
- **Anchorage and Berthing Guidelines:** the document outlines designated anchorage zones, permissible depths, and berthing constraints tailored to various vessel sizes and cargo types. It also includes information on restricted zones, such as those near pipelines or submarine cables, which are crucial for ensuring user safety and regulatory compliance (see Figure B.4).
- **Port Regulations and Reporting Protocols:** each port section provides information on specific regulatory requirements, including mandatory reporting for vessels above certain sizes or carrying hazardous materials (See for e.g. the regulations paragraph in Figure B.1). This enables the R.A.G. system to provide users with accurate compliance guidelines, reducing risks of penalties and enhancing port safety.
- **Safety Warnings and Restricted Areas:** the document highlights prohibited areas, including no-anchor zones, diving restrictions, and zones around historic wrecks or marine reserves (See caution paragraph in Figure B.4). By integrating this information, the R.A.G. system can help users avoid restricted or hazardous areas, promoting both safety and environmental preservation.
- **Contact information:** the PDF provides contact details for the harbor master's office, port authority, pilots or berthing authorities in a given port (see the table in Figure B.3). Depending on availability, it provides the telephone, e-mail, VHF channels or sometimes the website for these authorities or services.

In the document there are also some pictures of the bays or views of the ports and sometimes the information illustrated before is structured as a table, often the contact information is displayed as a table. The images aren't taken into consideration in this work because the SLMs used at the moment are not multi-modal large language models (MM-LLM) [CCB⁺24] but simpler text generation models ⁸.

⁸Multi-modal models have a backbone formed by an LLM and supplement this with the ability to take image and sometimes audio input in addition to text and then generate text or image output.

4.3 R.A.G

In this thesis (see element C in Figure 4.1), in order to split the external knowledge into chunks, a recursive text-splitter⁹ has been used as a text-splitter, thanks to its integration in Langchain, thought precisely only for generic text. This type of text-splitter could struggle to split the tables into coherent chunks and extract relevant information from them.

These problems could be solved with a careful part of pre-processing like extracting the tables¹⁰ and transforming them into normal text or cutting out the image or replacing it with a textual description using a MM-LLM¹¹ or with classical method for image captioning. All these steps could improve the quality of the information that can be retrieved from the document and then improve the final results of the whole chat-bot R.A.G system. These steps represent future developments of this work. For the following part of the test, the plain document without processing will be used as external knowledge.

4.3.1 Vector index

The first element to be created as part of an R.A.G system is the vector index. In this project, a library called *Facebook AI Similarity Search* [DGD⁺24] was used, normally known simply as **FAISS**, already integrated with LangChain¹², in its CPU-optimized version that exploits, for example, SIMD operations with AVX/AVX2 type instructions to speed up the execution times. Actually, this library is not dedicated to the creation of a classical vector database but to data indexing and thus creates a vector index (a sort of VecDBs without the possibility to carry out the classical CRUD operations) and performs similarity search between vectors. This is acceptable, because in the prototype it is not thought, for now, that for example a user can insert a new external knowledge source into the knowledge base. The knowledge base is considered static for reasons already explained in Section 4.2. The vector index should be created from a more powerful machine than the device edge and then moved in. To create the vector index the execution time is about 200 seconds using a normal laptop.

To create the vector index, we implemented a function called `create_vectordb`, which processes the external knowledge and constructs the vector index that will be used by the

⁹https://python.langchain.com/v0.2/api_reference/text_splitters/character.html#langchain-text-splitters-character

¹⁰Tools already exist for these types of problems: <https://docs.unstructured.io/welcome>, <https://www.aryn.ai/>

¹¹<https://medium.com/@akriti.upadhyay/unveiling-the-power-of-multimodal-language-models-in-image-captioning-970932de0e5f>

¹²FAISS component in LangChain

R.A.G system for retrieval. The function, shown below, follows several steps:

```

1 def create_vectordb(docs, spliter_name, chunk_size, chunk_overlap,
2     embedding_model, vector_store_name = vectorstore_path )-> str:
3     """
4         Create a vectorstore from a list of documents
5
6     Params:
7         docs: list of documents
8         spliter_name: text splitter name
9         chunk_size: chunk size
10        chunk_overlap: chunk overlap
11        vectorstore_path: path to save vectorstore
12    Returns:
13        vectorstore
14    """
15    documents = []
16    for doc in docs:
17        if type(doc) is not str:
18            doc = doc.name
19        documents.extend(load_single_document(doc))
20
21    text_splitter = TEXT_SPLITTERS[spliter_name](chunk_size=chunk_size,
22        chunk_overlap=chunk_overlap,)
23    texts = text_splitter.split_documents(documents)
24    start_time = time.time()
25    vector_index = FAISS.from_documents(texts, load_ov_embedding_model(
26        embedding_model))
27    print("--- %s seconds ---" % (time.time() - start_time))
28    vector_index.save_local(vector_store_name)
29
30    return "Vectorstore created at {}".format(vector_store_name)

```

The first step of the function is to gather each document in the list `docs` (In our case only an element). If the document isn't already a string, it is converted to a format that can be processed, and then loaded individually using the `load_single_document` auxiliary function that use the PyPDFLoader library to load the pdf files. Next, the loaded documents are split using the `text_splitter`. This component, previously discussed in Section 4.2, divides the documents based on the specified `chunk_size` and `chunk_overlap` parameters. The third step is to finally create the actual `vector_index` using the method `from_documents` provided by the FAISS library, which takes as argument the set of chunks prepared previously and the embedding model (we will see this component in detail in the next section) using the function `load_ov_embedding_model(embedding_model)` that we will see in the embedding model Section 4.3.2. Finally, the function saves the vector index to a specified path, `vector_store_name`, allowing it to be accessed and reused by the R.A.G system. A time-stamp message is printed to indicate the time taken for creating

and saving the vector index.

4.3.2 Embedding model

We have chosen a model developed by the Beijing Academy of Artificial Intelligence called **bge-small-en-v1.5**¹³ in its smallest version, which weighs 134 MB, obviously for reasons related to the few resources available in the edge domain. The model, has a BERT[DCLT19] type architecture with 24 Million parameters and supports only the English language.

An embedding model is not only used in the creation of the vectors for the vector index but also later to be able to compare the question posed to the chat-bot by the user with previously saved vectors and then return the most similar ones so as to create context. While it remains true, per se, that the vector index can be created on a more powerful machine capable of holding larger models and then move only the vector index to the edge device, it is equally true, however, that it is advisable to use the same model for searching, as discussed in [CDZ⁺24]. That is why we have chosen, since the creation of the vector index, a model small enough to then also be used for searching, on an edge device.

In order to enable the download and thus the use and reuse of the embedding model that is compatible with OpenVINO, we created the function `load_ov_embedding_model`. Here is the function and then how it works:

```
1 def load_ov_embedding_model(model_name_or_path: str = "BAAI/bge-small-en-  
2     v1.5") -> OpenVINOBgeEmbeddings:  
3     """  
4         Args:  
5             model_name_or_path (str): The name of the model to load.  
6         Returns:  
7             OpenVINOBgeEmbeddings: The loaded model.  
8         """  
9     try:  
10         logging.info("Loading openvino embedding model")  
11         model_kwargs = {'device': 'cpu'}  
12         encode_kwargs = {'mean_pooling': False, 'normalize_embeddings':  
13             True, "batch_size": 4}  
14         ov = OpenVINOBgeEmbeddings(  
15             model_name_or_path=model_name_or_path,  
16             model_kwargs=model_kwargs,  
17             encode_kwargs=encode_kwargs,  
18             show_progress=True,  
19             )  
20         return ov  
21     except Exception as e:
```

¹³Model card on HuggingFace

```
20     raise ValueError("Error loading openvino embedding model{}".  
format(e))
```

The function accepts a parameter, `model_name_or_path`, which specifies the embedding model's name or path to load. By default, it is set to “**BAAI/bge-small-en-v1.5**”. The function defines two parameter sets as dictionaries:

1. The first dictionary, `model_kwargs`, specifies options directly for the model. Here, the device is set to ‘cpu’ because the model will run on CPU.
2. The second dictionary, `encode_kwargs`, provides configuration for the encoding process, impacting how embeddings are generated. For instance, `mean_pooling` is set to False, which means that embeddings won’t be averaged over tokens. This configuration can be useful for certain models that may not require pooling to perform effectively. Another parameter, `normalize_embeddings`, is set to True, ensuring that the embeddings are normalized. Normalization standardizes the embeddings, making them consistent and more accurate for similarity comparisons. Finally, `batch_size` is defined as 4, allowing the model to process four inputs at a time, balancing efficiency with resource constraints, especially on a CPU.

The function then proceeds to load the embedding model using the class `OpenVINOBgeEmbeddings`, made accessible through LangChain. This class provides an optimized setup for OpenVINO-compatible embedding models. If the function is called for the first time, it handles both the downloading of the model specified by the `model_name_or_path` argument and the instantiation of the embedding model. If the model has already been downloaded to local storage, the function simply instantiates the model using the previously defined configurations. Finally, the function returns an instance of the `OpenVINOBgeEmbeddings` class ready to be used for creating the vector index or using the retriever.

4.3.3 Re-ranker model

One element, certainly not fundamental to the architecture of an R.A.G. system, but one that improves its performance is the so-called re-ranking [NC19] of context-entered documents found through the retriever. The idea is to use a new model to read and then sort the documents according to their importance with reference to the user’s query so that the SLM takes, for its response, the most important documents first. In the project has been used LangChain’s `OpenVINOReranker` class, also integrated with OpenVINO, to load and then use the model within the R.A.G pipeline by updating the documents found by the retriever.

The re-ranking model used here was released, like the model for the embedding part, by the Beijing Academy of Artificial Intelligence and is called **bge-reranker-v2-m3**¹⁴. This model, based on the bge-m3 [CXZ²⁴] model, is not too light with 568 million parameters. Therefore, to speed up the whole process it was compressed by quantization using the integration of NNCF [KLS²¹] in OpenVINO directly with the optimum-cli¹⁵ command to the precision INT8 of the original F32 (float with 32 bits).

Below is the function used to load and configure this re-ranking model:

```

1 def load_reranker_model(rerank_model_name: str = "../reranker/bge-
2   reranker-v2-m3_quantized") -> OpenVINOReranker:
3 """
4     Args:
5         rerank_model_name (str): The name of the model to load.
6     Returns: OpenVINOReranker: the loaded model.
7 """
8 try:
9     rerank_model_kwargs = {"device": "CPU"}
10    rerank_top_n = 2
11
12    reranker = OpenVINOReranker(
13        model_name_or_path=rerank_model_name,
14        model_kwargs=rerank_model_kwargs,
15        top_n=rerank_top_n,
16    )
17    logging.info("Loading reranker from -> {}".format(rerank_model_name))
18    return reranker
19 except Exception as e:
20     raise ValueError("Error loading reranker{}".format(e))

```

The `load_reranker_model` function is responsible for loading the re-ranking model in a way that makes it compatible with the R.A.G pipeline. Here's how it works:

1. Model Configuration:

- The function begins by setting the `rerank_model_kwargs`, which specifies the device to be used as 'CPU', ensuring compatibility with local deployments.
- The `rerank_top_n` variable is set to 2, meaning the model will prioritize the top two documents for re-ranking. This parameter determines how many documents will be evaluated for the final re-ordering, balancing between efficiency and accuracy.

2. Loading the Re-Ranker Model:

¹⁴HuggingFace's model card

¹⁵<https://huggingface.co/docs/optimum/main/en/intel/openvino/export>

- Using `OpenVINOReranker`, the function loads the specified model with the configurations provided. The `model_name_or_path` argument specifies the path to the quantized model, and the `top_n` argument applies the `rerank_top_n` setting.

By returning an `OpenVINOReranker` instance, this function makes the re-ranking model easily accessible within the R.A.G pipeline, enhancing the relevance of retrieved documents before they are passed to the SLM for response generation. As with the previous functions, the model will be used directly if it has already been downloaded, otherwise it needs to be downloaded and thus an Internet connection. This setup ensures that the most relevant documents are prioritized, improving the accuracy and reliability of the chat-bot's responses.

4.4 MarineChat... Assemble!

Following a detailed examination of the constituent elements of the R.A.G. pipeline, including the vector index, re-ranker, embedding model and SLM initialization, the objective now is to integrate these components into a fully operational R.A.G.-based chat-bot (see Figure 4.1 element A). This section will demonstrate how the preceding elements have been assembled to create MarineChat.

The core of this integration is the '`create_rag_chain`' function, which unites the vector index, retriever, re-ranker, and language model to form a unified pipeline. The prototype of the function is provided below.

```
1 def create_rag_chain(vector_index, slm, vector_search_top_k,
2     vector_rerank_top_n, reranker, search_method, score_threshold,
3     prompt_template, default_rag_prompt = "QWEN_DEFAULT_RAG_PROMPT"):
4     """
5         Create a RAG chain from a vectorstore
6
7     Params:
8         db: vectorstore
9         llm: language model
10        vector_search_top_k: top k for search
11        vector_rerank_top_n: top n for rerank
12        run_rerank: run rerank
13        search_method: search method
14        score_threshold: score threshold
15
16    Returns:
17        RAG chain
18    """
19    if vector_rerank_top_n > vector_search_top_k:
```

```

17     raise ValueError("Search top k must >= Rerank top n")
18
19     if search_method == "similarity_score_threshold":
20         search_kwargs = {"k": vector_search_top_k, "score_threshold": score_threshold}
21     else:
22         search_kwargs = {"k": vector_search_top_k}
23     retriever = vector_index.as_retriever(search_kwargs=search_kwargs, search_type=search_method)
24     if reranker:
25         reranker.top_n = vector_rerank_top_n
26         retriever = ContextualCompressionRetriever(base_compressor=reranker, base_retriever=retriever)
27     prompt = PromptTemplate(input_variables=[default_rag_prompt, "context", "question"], template=prompt_template)
28     combine_docs_chain = create_stuff_documents_chain(slm, prompt)
29     return create_retrieval_chain(retriever, combine_docs_chain)

```

4.4.1 Explanation of create_rag_chain

The `create_rag_chain` function serves as the central component of MarineChat, integrating the various elements of the R.A.G. pipeline into a single chain that handles retrieval, re-ranking, and response generation. In the next sub-sections we will look in detail at how the function works.

4.4.2 Validation of Parameters

The function begins by ensuring that the number of documents to be re-ranked (`vector_rerank_top_n`) does not exceed the number of documents retrieved (`vector_search_top_k`). This check is crucial because re-ranking can only operate on documents that have already been retrieved. If this condition is violated, the function raises a `ValueError` to prevent logical inconsistencies in the pipeline.

4.4.3 Configuring the Retriever

The retriever is constructed from the index vector using the FAISS's `as_retriever` method that takes two arguments: `search_type` and `search_kwargs`. Depending on the `search_method` parameter, the function sets up the retriever's search behavior initializing the `search_kwargs` dictionary, then pass it as parameter with the `search_method` string to the retriever method. If "`similarity_score_threshold`" is specified, the retriever uses a minimum

relevance threshold (`score_threshold`) to filter documents in addition to retrieving up to `vector_search_top_k` results. For other methods, the retriever simply limits the results to the top `vector_search_top_k` most relevant documents. This flexibility allows the pipeline to adapt to different retrieval strategies. For the testing phase, the "`similarity_score_threshold`" search method was employed, and a median score threshold of 0.6 was set.

4.4.4 Integrating the Re-Ranker

If a re-ranker object is provided (`reranker` parameter), the function integrates it using the `ContextualCompressionRetriever`. This LangChain's component wraps the base retriever and applies the re-ranker as a compression model. The retrieval process becomes a two-stage pipeline:

- In the first stage, the base retriever fetches an initial set of documents based on vector similarity.
- In the second stage, the `ContextualCompressionRetriever` filters these documents, retaining only the top `vector_rerank_top_n` documents deemed most relevant by the re-ranker. The remaining documents are discarded and are no longer available for further processing.

This ensures that only the most relevant documents are passed to the language model, optimizing the context provided for response generation. This technique is designed to ensure that important information is not buried under a large volume of less relevant content. For models with limited context lengths, such as the gemma 2B model 3.2.2, it enables the creation of a usable context that stays within the model's constraints. More generally, this method simplifies the input for all SLMs by reducing the complexity of overly long contexts, making it easier for the models to focus on the most relevant information and generate more accurate and coherent responses.

4.4.5 Constructing the Prompt

To enable the SLM to generate meaningful responses, the function creates a `PromptTemplate` class. This class defines the structure of the prompt passed to the language model, including placeholders for the default R.A.G. prompt, the retrieved context, and the user's question. This modularity allows the prompt to be easily adapted for different use cases and different models. Since each family of SLMs typically requires a specific system prompt template to perform optimally, selecting the appropriate prompt is crucial to obtaining

the best possible responses. This can be done by using the parameters `prompt_template` and `default_rag_prompt`, changing them according to the SLM you have decided to use, this will then be an important element to carry out the tests considering different models. Here's below an example of a `prompt_template`. This is the one used for the Qwen 2.5 series models.

```

1 qwen_rag_prompt_template = f"""
2     {QWEN_DEFAULT_RAG_PROMPT }<|im_start|>system
3     <|im_start|>user
4     Question: {input}
5     Context: {context}
6     <|im_end|>
7     <|im_start|>assistant<|im_end|>
8 """

```

The correct template format in the case of the Qwen 2.5 series was simply not provided within the template card as it is for other models. So we found it and extracted it from the `tokenizer_config.json` file¹⁶, which is located within the various files that make up the model. In this snippet of code, the placeholders `{input}` and `{context}` are used within the `prompt_template` to dynamically incorporate the user's query and the retrieved documents, respectively. The `PromptTemplate` class ensures that each passed element appears in the right placeholder.

Special tokens such as `<|im_start|>` and `<|im_end|>` are used in the Qwen 2.5 series models to delimit sections within the prompt. These tokens signal the start and end of different input roles (e.g., system instructions, user queries, and assistant responses), enabling the model to understand and generate responses appropriately. For example:

- `<|im_start|>system` indicates the beginning of the system instructions (e.g., general guidelines for the assistant's behavior).
- `<|im_start|>user` signals the start of the user's input.
- `<|im_start|>assistant` prepares the model for generating a response.

These tokens are crucial for guiding the model's behavior and ensuring it distinguishes between the various roles in the conversation. Without these delimiters, the model might misinterpret the prompt, resulting in less accurate or contextually incorrect responses. The other two different `prompt_template` instances, one for each of the remaining model families considered in this work, along with the two `default_rag_prompt` configurations that were written and used, are listed in the Appendix C for clarity.

¹⁶https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct/blob/main/tokenizer_config.json

4.4.6 Combining Documents and Query

The retrieved and re-ranked documents, along with the user's query and the SLM, are passed to the function `create_stuff_documents_chain`. This function constructs a processing chain that combines the retrieved context with the user's query into the prompt defined earlier, then passes that prompt to the SLM, ensuring that the language model (SLM) receives a clear and concise input.

4.4.7 Returning the R.A.G. Chain

The function concludes by using `create_retrieval_chain` to assemble the complete R.A.G. pipeline. This step combines the configured retriever (which may include optional re-ranking through the `ContextualCompressionRetriever`) with the document combination chain created earlier. The resulting R.A.G. chain is a fully functional pipeline capable of performing three key tasks:

- Retrieving relevant documents from the vector store based on the user's query.
- Refining the results through re-ranking, if enabled, to prioritize the most relevant context.
- Generating a final response by processing the query and retrieved context through the language model.

By assembling all the components into a single pipeline, the `create_rag_chain` function encapsulates the functionality required to handle user queries effectively. The retriever ensures that only relevant documents are considered, the re-ranker refines the document ordering, and the SLM generates responses that are both coherent and informed by the retrieved context. This modular and efficient design makes MarineChat a robust tool for addressing complex queries in the maritime domain.

4.5 Implementation of the Chat-bot

With all the components of the R.A.G. pipeline in place and the SLM initialized, the final step is to implement the chat-bot itself. This implementation acts as the interface that interacts with the user, processes their queries, retrieves relevant information and generates contextually accurate responses. Another slightly different version has been implemented to automate the testing part, but only the user-friendly version of the chat-bot code is reported below:

```

1 try:
2     # Take the input from the user inside a loop to keep the chatbot
3     # running
4     while True:
5         query = input("Insert here your question (type exit to quit): ")
6         if query == "exit":
7             if output:
8                 request_cancel(ov_slm=ov_slm)
9             break
10        start = time.time()
11        output = rag_chain.invoke(input={"input": query})
12        print("Time taken: ", time.time() - start)
13        print(output['answer'])
14        print("-" * 100) # Separator between each question/answer
15        request_cancel(ov_slm=ov_slm)
16
17 except KeyboardInterrupt:
18     print("Session ended.")
19 finally:
20     del vectorstore
21     del reranker
22     del ov_slm
23     print("Resources released.")

```

The chat-bot (see element A in Figure 4.1) is implemented as a very simple interactive Python script that runs in a loop, continuously processing user queries until explicitly terminated. Below is a breakdown of the code:

1. **Interactive Query Loop:** The script begins with a `while` loop that allows users to enter queries. The user is prompted with the message `"Insert here your question (type exit to quit):"`. If the user types `exit`, the chat-bot exits the loop and proceeds to release resources.
2. **Query Processing:** For each query entered by the user:
 - The start time is recorded using `time.time()` to measure the response time.
 - The query is passed to the R.A.G. chain using the `rag_chain.invoke` method. This method processes the query by retrieving relevant information, combining it with the context, and generating a response using the SLM.
 - The response is extracted from the output dictionary (`output['answer']`) and printed to the console for the user.
3. **Session Termination and Resource Management:**
 - The chat-bot gracefully handles termination via two mechanisms:

- If the user types `exit`, the chat-bot releases resources and exits the loop.
- If the user interrupts the script with `KeyboardInterrupt` (e.g., pressing `Ctrl+C`), the session ends, and the resources are released in the `finally` block.
- In both cases, the resources used by the chat-bot (`vectorstore`, `reranker`, and `ov_slm`) are explicitly deleted using `del`. This ensures proper cleanup of memory and prevents resource leaks, especially in long-running sessions. This part was added to try to figure out how to handle a non-deterministic and still open problem on the handling of SLM by OpenVINO that was repeatedly encountered during the implementation of the tool. This part was added to try to figure out how to deal with an apparently non-deterministic and still open problem¹⁷ on the handling of SLM processes by OpenVINO that was repeatedly encountered during the implementation of the tool.

This implementation provides a robust and user-friendly interface for interacting with MarineChat. The modularity of the pipeline, combined with the simplicity of the query loop, ensures that the chat-bot is both efficient and extensible¹⁸.

¹⁷The problem reported as an issue on the OpenVINO github repo (date 15/11/2024).

¹⁸At this link the Github repository with our entire project shown in this chapter

Chapter 5

Experimental results

The intent of this thesis is not to find the right balance to get the best possible chat-bot on edge devices by trying various combination of many features which can change the performance of one part or the other of the system such as: regarding the R.A.G part, the chunk size, the chunk overlap , different types of text splitters, search methods in the vector index or different thresholds or even for the chat-bot part model temperature,`top_n`, `top_k` and so on. All this can be seen as an interesting future development of this work.

In this Chapter we discuss if the results are satisfactory given the resource constraints imposed by the edge computing environment.

We will focus on how these results can be attributed to different usable models in our environment, we will test on the same evaluation set different SLMs of the families already presented in the Chapter 3, also trying different levels of quantization for the largest ones. It must be specified here, that the testing of chat-bots with R.A.G is still an open problem in the literature [YGZ⁺24]. Many methods have been presented like ARES ¹ [SFKPZ23], RAGAs ² [EJEAS23], RGB ³ [CLHS24] and AutoRAG ⁴ [FQY⁺24] there is no commonly accepted solution.

¹<https://github.com/stanford-futuredata/ARES>

²<https://github.com/explodinggradients/ragas>

³<https://github.com/chen700564/RGB>

⁴https://fujia19990407.github.io/project_website/

5.1 Hardware configuration

5.1.1 CPU

As a first use case, we tried to use a personal computer (a Lenovo ThinkBook 14 G7 IML) and exploiting its CPU for the inference of the SLM or embedding model. This corresponds to the edge device in Section 3.1.2, because the features are: a Intel® Core™ Ultra 7 Processor 155H, with a 16 GB of installed DDR5-5600 RAM. It has an energetic consumption for the processor of 28 W and costs around 1000€⁵.

To take advantage of this capability, we developed the R.A.G chat-bot with OpenVINO™⁶ integrated with the LangChain and HuggingFace libraries. Of course, despite the difficulties, not only does OpenVINO™ exist to efficiently run AI models on CPUs but other open-source libraries have also arisen such as perhaps the most famous llama.cpp⁷. So our choice fell on OpenVINO™ so that we could explore a framework designed specifically for the hardware at our disposal.

5.1.2 GPU

Announced on March 18, 2019, at the GPU Technology Conference (GTC) in San Jose, California, NVIDIA presented the Jetson Nano, positioning it as an affordable edge computing platform designed for running machine learning and AI models.⁸ Before the introduction of the Jetson Nano 2GB in 2020⁹, which further reduced costs, the Jetson Nano was the most affordable GPU platform offered by NVIDIA.

The Jetson Nano features a quad-core ARM Cortex-A57 processor operating at 1.43 GHz, paired with a 128-core Maxwell GPU and it's equipped with 4GB of LPDDR4 RAM¹⁰. It performs 472 GOPS with an energy consumption between 5w and 10W¹¹.

The JetPack SDK, built on CUDA-X, comprising over 40 GPU-acceleration libraries, includes optimized libraries for deep learning, supporting the entire Jetson family. JetPack also provides the latest versions of CUDA, cuDNN, TensorRT™, and a complete desktop Linux operating system.

⁵https://psref.lenovo.com/Product/ThinkBook/ThinkBook_14_G7_IML

⁶In this thesis was used the 2024.3 version

⁷ Llama.cpp'Github repository

⁸<https://www.forbes.com/sites/janakirammsv/2019/03/19/nvidia-launches-jetson-nano-a-competitor-to-google-coral-and-intel-up-squared/>

⁹<https://nvidianews.nvidia.com/news/nvidia-unveils-jetson-nano-2gb-the-ultimate-ai-and-robotics-starter-kit-for-students-educators-robotics-hobbyists>

¹⁰<https://developer.nvidia.com/embedded/jetson-nano>

¹¹<https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/product-development/>

In 2021 NVIDIA announced that the Jetson Nano will no longer receive updates for the JetPack SDK, concluding its compatibility with version 4.6.5.

¹² The lack of support for newer versions of CUDA and other key libraries, like Python and Pytorch ¹³, severely limits the Jetson Nano's capabilities, particularly in running language models and the use of key libraries like Hugging Face.

Given the limitations for libraries, the Jetson Nano has become impractical for running contemporary AI projects.

At the beginning of the work, together with BSc. Zanetta, we had started to use this device, but after many trials, for the reasons just outlined, we had to change our plans and switch, to a more powerful Jetson device, the Jetson AGX Xavier.

Released a year before of the Jetson Nano, on September 12, 2018 ¹⁴, NVIDIA announced the release of the Jetson AGX Xavier, a high-performance edge AI computing platform, designed primarily for robotics and autonomous machines.

The Jetson AGX Xavier is powered by an 8-core NVIDIA Carmel 64-bit ARMv8.2 CPU and a 512-core NVIDIA Volta GPU, equipped with 64 Tensor Cores capable, it includes 16GB of 256-bit LPDDR4x memory and 32GB of eMMC 5.1 storage ¹⁵. It is capable of 32 TOPS at energy consumption between 10W to 30W.

One of the standout architectural features of the Jetson AGX Xavier is its unified memory architecture, which allows the CPU and GPU to share the same memory. Unlike conventional systems where the GPU requires separate memory and must rely on memory transfers, this design improves on these bottlenecks as well as reduced power consumption ¹⁶.

As with other members of the Jetson family, the AGX Xavier is supported by JetPack SDK. Unlike the Jetson Nano, which is limited to JetPack 4.6.5, the AGX Xavier benefits from extended support, including compatibility with JetPack 5.1.4. ¹⁷ This version incorporates more recent libraries, such as CUDA 11.4, expanding the platform's capabilities. This enhanced compatibility allows the use of tools such as LangChain and Hugging Face libraries, but in non-up-to-date versions.

¹²<https://forums.developer.nvidia.com/t/jetson-software-roadmap-for-2h-2021-and-2022/177724>

¹³<https://developer.nvidia.com/jetpack-sdk-465>

¹⁴https://developer.nvidia.com/blog/nvidia-jetson-agx_xavier-developer-kit-now-available/

¹⁵<https://www.nvidia.com/it-it/autonomous-machines/embedded-systems/jetson-agx-xavier/>

¹⁶<https://www.forbes.com/sites/patrickmoorhead/2018/12/17/robotics-and-smart-cities-just-got-a-lot-more-real-with-nvidia-jetson-xavier-edge-compute-module/>

¹⁷<https://developer.nvidia.com/embedded/jetpack-archive>

5.2 Evaluation set

One of the few certainties is that in order to test a system such as ours we need a specific evaluation set concerning our use case and containing on the one hand the questions of the hypothetical user and on the other hand the “right” answers (what is definable as right actually is in itself another problem) that we expect to receive from the system. It can be easily guessed, creating such a set questions and answers is very challenging, primarily because highly specific domains, such as the maritime one, require uncommon knowledge not only to formulate accurate questions but, even more importantly, to answer them correctly. To deal with this problem, the most natural solution is to find an expert or a group of experts in the field and use them to validate questions and answers. This solution is the best, but it requires finding experts and, resources to pay them. For these reasons, many attempts have been made to find other systems of evaluation and many works have gone at least partially in this direction. For example, the RAGAs method previously mentioned in Section 5.4 allows a partial evaluation of an R.A.G. system without expert validation; we will look at this method in detail in the next section.

Unfortunately, for this project it was not possible to find experts willing to create a set of user’s queries and correct answers as explained above. We contacted Dr. Fabio Parodi of the Tecnoleader company to participate as an expert, given his general maritime knowledge and his role in a company in the same sector, but due to work commitments he was unable to accept.

Furthermore, to the best of our knowledge, there are no existing evaluation set for the use case of this project, so it was decided to create a small one from scratch, including, in addition to the questions of course, also the expected **correct answers** (also called ‘**ground_truths**’ or ‘**reference**’ in the tool which we will be presented in the Section 5.4). For this purpose, the author’s limited knowledge of maritime topics, specifically ports and navigation around the Italian peninsula, was utilized. So the questions are simple, the kind that the average small/medium boat sailor might ask. Aligning exactly with our use case. It is necessary to specify here that the questions, and therefore the answers, must be closely related to what is reported in the external dataset (see Section 4.2), in order to test, above all, the retrieval capabilities of the information. Therefore, it was considered that even without an expert, it would be possible to obtain a suitable set of data to evaluate the system, without taking advantage of the possibility of evaluating the system only with the questions without **ground_truth** as mentioned above.

The evaluation set consisted of **25 pairs of questions and ground truth**. The whole dataset can be seen in the Appendix A. The queries try to be geographically spread to cover all parts of the Italian coastline. At the same time, the questions have been designed not to be redundant and to take into account as many different maritime aspects as possible, in terms of information or problems, both general and more specific.

Let us now look briefly at some examples of questions and answers found in the evaluation set. Take for example the first question of the evaluation set and corresponding ground truth: *What are the typical winds in Genoa ? The typical winds in Genoa are in winter from N (tramontana) to NW while in summer from SE (Scirocco) and SW (Libeccio).*

The question is a simple one, but it is always important for sailors to know what the most common winds are in an area, especially if you are perhaps planning to dock in a harbour. The answer gives the essential details while remaining concise. The question was chosen because it is based on our personal experience. The second one pair of question and ground truth is: *Is there a VHF channel to contact for information about the passage of the Strait of Messina ? The VHF channels are 10, 13, 14, and 16.* This time, the question concerns the passage through the Strait of Messina, that is always a challenge for any sailor due to the strong tidal currents. The response is very clear, simple and concise.

5.3 Model tested

The evaluation set illustrated in the Appendix A was used (we will later see in detail how, including which metrics were adopted) using various SLMs, all belonging to the families we discussed in Chapter 3. As mentioned earlier, the larger models were quantized to improve inference speed (see Section 2.5). As far as language models are concerned, only SLMs have already been quantized, in order to avoid quantization of LLM to such an extent that they become small. To compress through quantization, we utilized the integration of OpenVINO¹⁸ with Huggingface’s Optimum library, which uses the NNCF compression engine [KLS⁺21]. The quickest and most direct way to do this is to use Optimum’s Command Line Interface (Optimum CLI), which allows you to download and compress a model with only a command. An example is shown below:

```
1 optimum-cli export openvino --model microsoft/Phi-3.5-mini-instruct
2 --task text-generation-with-past --weight-format int8
3 --trust-remote-code ../model/microsoft/Phi-3.5-mini-instruct/int8
```

Below, we briefly see a description of this command.

- **optimum-cli export openvino:** This specifies that you are using the Optimum-CLI tool to export a model in a format compatible with OpenVINO for optimized inference.
- **--model microsoft/Phi-3.5-mini-instruct:** Specifies the model to be downloaded and optimized. In this case, the model identifier is **microsoft/Phi-3.5-mini-instruct**, which is hosted on Hugging Face.

¹⁸<https://docs.openvino.ai/2024/openvino-workflow/model-optimization.html>

- `--task text-generation-with-past`: Indicates the task type for which the model is being optimized. For text generation tasks, the past key values are used to store previously computed attention scores, speeding up token generation in sequences.
- `--weight-format int8`: Specifies the target quantization format, reducing the model weights to 8-bit integers (INT8). This compression method greatly reduces model size and improves inference speed while maintaining acceptable accuracy levels.
- `--trust-remote-code`: Enables execution of custom code that may be provided by the model’s repository (e.g., custom layers or components). This is necessary if the model requires custom implementations not available in the default libraries.
- `../model/microsoft/Phi-3.5-mini-instruct/int8`: Specifies the output directory where the optimized and quantized model will be saved.

The quantized models are Phi-3.5 mini with 3.8B¹⁹ parameters and Qwen2.5 with 3B parameters, as shown in Table 5.1 below. The latter was quantized by reducing the weight precision to INT8. Meanwhile, the first model was also quantized with asymmetric²⁰ INT4 precision²¹. All other models will be used without any compression, because we assume that given their size, the inference time is still acceptable without quantization and at the same time would lower their capabilities too much.

No Mixture of Expert models²² (which fall under the category of dynamic networks mentioned in Section 2.5) were used due to the inability to download them directly using the previously mentioned libraries. This remains an avenue to explore in the future.

Vendor/Name	Size (b)	Size (GB) BQ	Size (GB) AQ	Quantization
microsoft/Phi-3.5-mini-instruct	3.82 b	7.11	2.28/ 3.56	INT4 ASYM - INT8
Qwen/Qwen-2.5-0.5B-Instruct	0.494 b	0.953	0.953	None
Qwen/Qwen-2.5-1.5B-Instruct	1.54 b	2.88	2.88	None
Qwen/Qwen-2.5-3B-Instruct	3.09 b	5.75	2.92	INT8
google/gemma-2-2b-it	2.61 b	4.88	4.88	None

Table 5.1: The different models tested with their size in number of parameters the type of quantization and the size in GB before (BQ) and after quantization (AQ). The letter b stands for billion and size (b) means the size per billion parameters.

¹⁹The letter B stands for billions. It is a way normally used to indicate the number of parameters of a language model. It will also be used in this thesis for brevity.

²⁰INT4 Asymmetric mode uses an unsigned 4-bit integer to quantizes weights asymmetrically with a non-fixed zero point. This mode slightly compromises speed in favor of better accuracy compared to the symmetric mode.

²¹The group size was set to 128, and the ratio to 0.8 <https://docs.openvino.ai/2024/openvino-workflow/model-optimization-guide/weight-compression.html>

²²<https://huggingface.co/blog/moe>

5.4 Evaluation Metrics

Alongside the execution times, we will evaluate the “goodness” of the results using the *Retrieval Augmented Generation Assessment* framework, in short RAGAs²³ [EJEAS23]. It is a flexible tool already integrated with LangChain, designed to test both the R.A.G pipeline and the response of the SLM. It allows you to calculate, four main metrics: **faithfulness**, **response relevancy**, **context precision** and **semantic similarity**. With the column `ground_truth` (thus annotation by a human), called in RAGAs `reference`, two other metrics referring to R.A.G. namely **context recall** and **factual correctness** can also be calculated²⁴. In order to obtain the metrics, RAGAs needs, in addition to the elements already present in the initial evaluation set (see Section 5.2), the response provided by the tool and the context retrieved by the R.A.G part of the system. It is therefore necessary to obtain an **evaluation data set**, and then to pass it to the framework, which has exactly these column names (see Appendix D):

1. `user_input`: this column represents the user’s question. In the initial dataset is called simply `question`
2. `retrieved_contexts`: he retrieved context from the external knowledge base
3. `response`: the final answer generated by the SLM
4. `reference`: the reference answer. What was called `ground_truth` in the initial evaluation set

To calculate these metrics, RAGAs uses an external large language model, which can be one of a number of user choices. By default, RAGAs is designed to use Chat-GPT. Only semantic similarity does not require an LLM to be computed, but instead requires an embedding model such as the ones we discussed earlier in Section 4.3.2 . Let us now see what these six metrics mean and how they are calculated.

Faithfulness: This metric estimates whether the statements present in the response generated by the R.A.G chat-bot are actually supported by the retrieved context. The goal is to ensure that the response is based on factual information present in the context and not on hallucinations or inventions of the SLM. The answer is scaled to [0,1] range, the higher the better. The generated answer is regarded as faithful if all the claims made in the answer can be inferred from the given context. To calculate this, a set of claims from the generated answer is first identified. Then each of these claims is cross-checked with the

²³The Github repo for the RAGAs project

²⁴These are the only metrics used by the most recent version of the framework

given context to determine if it can be inferred from the context. The faithfulness score is given by:

$$\text{Faithfulness score} = \frac{\text{Number of claims in the generated answer that can be inferred from the context}}{\text{Total number of claims in the generated answer}}$$

Response relevancy: This second metric focuses on assessing how pertinent the generated answer is to the given prompt. A lower score is assigned to answers that are incomplete or contain redundant information and higher scores indicate better relevancy. This metric is computed using the `user_input`, the `retrieved_contexts` and the response. The Answer Relevancy is defined as the mean cosine similarity [SEK24] of the original `user_input` to a number of artificial questions, which were generated (reverse engineered) by the LLM based on the response:

$$\text{answer relevancy} = \frac{1}{N} \sum_{i=1}^N \cos(E_{g_i}, E_o)$$

Where:

- E_{g_i} is the embedding of the generated question i .
- E_o is the embedding of the original question.
- N is the number of generated questions, which is 3 by default.
- $\cos(E_{g_i}, E_o)$ stands for the computation of cosine similarity

The underlying idea of this metric is that if the generated answer accurately addresses the initial question, the LLM should be able to generate questions from the answer that align with the original question. This metric can be considered the evolution of the Answer relevancy metric presented initially in the RAGAs's paper [EJEAS23].

Context precision without a reference is a metric that measures the proportion of relevant chunks in the `retrieved_contexts`. It is calculated as the mean of the precision@k for each chunk in the context. Precision@k is the ratio of the number of relevant chunks at rank k to the total number of chunks at rank k , where k is the total number of chunks in the `retrieved_contexts`.

$$\text{Context Precision@K} = \frac{\sum_{k=1}^K (\text{Precision}@k \times v_k)}{\text{Total number of relevant items in the top K results}}$$

$$\text{Precision}@k = \frac{\text{true positives}@k}{(\text{true positives}@k + \text{false positives}@k)}$$

Where k is the total number of chunks in `retrieved_contexts` and $v_k \in \{0, 1\}$ is the relevance indicator at rank.

In particular in this case we used a version of contextual precision called `LLMContextPrecisionWithoutReference` which, as the name suggest does not need the `ground_truth/reference` as information in the dataset. To estimate if a retrieved contexts is relevant or not this method uses the LLM to compare each chunks present in `retrieved_contexts` with response generated by the SLM. This decision is motivated by the fact that we try to avoid using the `ground_truth` or reference to calculate metrics.

Context recall evaluates how many of the relevant documents (or pieces of information) were successfully retrieved. It focuses on capturing all relevant information. Higher recall means fewer relevant documents were left out. In short, context recall is about not missing anything important. Since it is about not missing anything, calculating context recall always requires a reference to compare against.

To compute this metric we have used the `LLMContextRecall`, which uses the `user_input`, `reference` and the `retrieved_contexts`, and the values range between 0 and 1, with higher values indicating better performance. This metric uses reference as a proxy to `reference_contexts` which also makes it easier to use as annotating reference contexts can be very time consuming. To estimate context recall from the reference, the reference is broken down into claims each claim in the reference answer is analyzed to determine whether it can be attributed to the retrieved context or not. In an ideal scenario, all claims in the reference answer should be attributable to the retrieved context.

The formula for calculating context recall is as follows:

$$\text{context recall} = \frac{\text{GT claims that can be attributed to context}}{\text{Number of claims in GT}}$$

where GT stands for Ground Truth.

Factual Correctness compares and evaluates the factual accuracy of the generated `response` with the `reference`. This metric is used to determine the extent to which the generated `response` aligns with the `reference`. The factual correctness score ranges [0,1], with higher values indicating better performance. To measure the alignment between the response and the reference, the metric uses the LLM for first break down the response and reference into claims and then uses natural language inference to determine the factual overlap between the response and the reference. Factual overlap is quantified using precision, recall, and F1-score, which can be controlled using the `mode` parameter, which is set to F1-score by default.

The formula for calculating True Positive (TP), False Positive (FP), and False Negative (FN) is as follows:

True Positive (TP) = Number of claims in response that are present in reference

False Positive (FP) = Number of claims in response that are not present in reference

False Negative (FN) = Number of claims in reference that are not present in response

The formula to calculate the F1-score is the following:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

We decided to keep the F1-score calculation because we can represent both precision and recall with only one metric. To show the trade-off between the other two metrics.

Semantic similarity: the concept of this metric pertains to the assessment of the semantic resemblance between the generated answer and the ground truth. This evaluation is based on the ground truth and the answer, with values falling within the range of 0 to 1. A higher score signifies a better alignment between the generated answer and the ground truth. Measuring the semantic similarity between answers can offer valuable insights into the quality of the generated response. It shows whether the generated response is semantically similar to ground truth. Unlike factual-correctness, this metric does not take into account the actual correctness of the answer. This evaluation utilizes a cross-encoder model to calculate the semantic similarity score.

Three steps are required to calculate the semantic similarity of answers:

1. *Vectorize the ground truth via the embedding model.*
2. *Vectorize the generated answer using the same embedding model.*
3. *Compute the cosine similarity between the two vectors.*

We have now seen what each metric represents and how it is calculated. In the following sections, we will see the results of calculating these metrics on our tests.

5.5 Test on CPU

In this section, we will present the tests performed on the Intel CPU version. We will then see the results of the calculation of the metrics and the inference times of the tool.

MarineChat was tested on the 25 question instances of the initial dataset, using the different SLMs considered to generate the answer and using the appropriate prompt template for

each one. Below, we see an example of the response given in this case by the model `Qwen/Qwen2.5-3B-Instruct`. We mention here the fact that given the nondeterminism of SLMs and language models in general [LLY22], [OZHW23] if the question is asked again to the same model we cannot expect the exact same answer.

```

1 Prompt:
2 You are a knowledgeable assistant specializing in answering questions
   about ports, sailing, and maritime activities along the Italian
   peninsula.
3 Use the most relevant pieces of the retrieved context below to support
   your answer.
4 If you don't know the answer or the context is insufficient, state that
   you are unsure.
5 Use a maximum of three sentences, and keep your answer concise, accurate,
   and where applicable, use maritime terminology.
6
7 Question: What are the typical winds in Genoa?
8
9 Context:
10 French border) to Genova, then SE and S to Promontorio di Piombino and
    Arcipelago Toscano.
11 General Remarks
12 10.1 Winds-Weather. In the Ligurian Sea and Golfe di Genova, the
    prevailing wind is from the NW practically all year.
13 However, locally along the coast, there is some variation due to the land
    configuration.
14
15 Response:
16 Based on the provided context, the prevailing wind in the Ligurian Sea
    and Gulf of Genova is from the northwest throughout the year.
17 However, local variations can be observed along the coast due to the land
    's configuration.
18 Near Genoa, winds typically blow from the southwest to the south, while
    further east towards the Tuscan Islands, they shift to the southeast.

```

Listing 5.1: MarineChat Prompt and Response

Starting from answers like the one ²⁵ above, we created an evaluation set for each model to be evaluated using RAGAs ²⁶. To do this, we extracted for each answer the retrieved context and the actual response.

At this point, the only thing that remains to be done is to calculate the six metrics set out above using the five evaluation sets, one for each model tested, properly processed. Two different methods were used to calculate these metrics. In the first case, it was decided to use an open model, specifically `meta-llama/Meta-Llama-3.1-70B-Instruct-Turbo`

²⁵It should be noted that there are no special tokens that are not of interest during evaluation

²⁶The link to the Google Drive folder containing all the files created.

²⁷ as LLM and `togethercomputer/m2-bert-80M-8k-retrieval` ²⁸ as embeddings model using the Together AI API service, and to calculate the metrics five times for each question for each different SLM used. The language model was chosen because it offered the best balance between performance and cost. The embedding model was chosen because it was the first to have a context window large enough to process the input at once.

Finally, we calculated the average of each metric for each answer. We came up with this solution to try and generate more answers and then get an aggregate number, as the result dependeds on a model that was not considered to be one of the best SLMs available.

On the other hand, the second method uses RAGA’s default model, `Chat-GPT 4o`, through the use of the OpenAI API, which is usually considered to be the best available LLM ²⁹ [SLM⁺²⁴] and `text-embedding-3-small` as embedding model because is the best in terms of trade-off between performance and cost ³⁰. In this case, we simply calculated the 6 metrics for each response generated by the different SLMs.

5.5.1 Overall quality evaluation

This section provides a general view of the results of the metrics for each model used, keeping in mind that the R.A.G system remains the same as the generative models change. To do this, we first consider the average of the metrics calculated by repeating the metrics calculation 5 times based on `Llama 3.1 70b` as the evaluation LLM and `m2-bert-80M-8k-retrieval` embedding model. Thus, metrics with a relatively worse LLM. We might expect metrics that are therefore not too reliable (which is why we calculated them 5 times each). In any case, below we can see a bar chart (Figure: 5.1) showing the results. Each bar represents a different SLM, with different responses from the others, resulting in different values for the metrics. The different metrics are shown on the x-axis.

²⁷Models marked as turbo like ours are quantized to FP8. It is also interesting to note that Together AI recommends an SLM for those who do not know which model to use.

²⁸<https://docs.together.ai/docs/serverless-models#embedding-models>

²⁹<https://lmarena.ai/?leaderboard>

³⁰<https://platform.openai.com/docs/guides/embeddings>

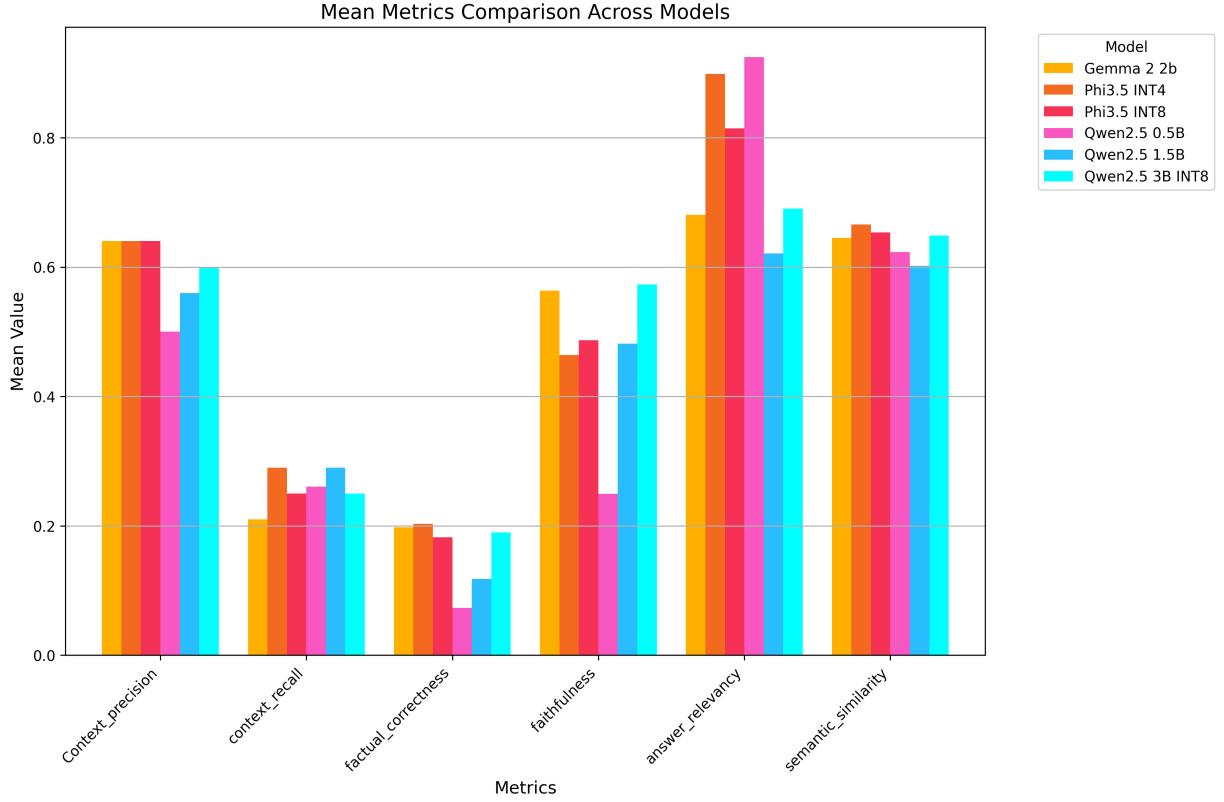


Figure 5.1: The metrics means over all the models using LLaMA 3.1 70b via TogetherAI API

From the plot, several trends can be observed. Faithfulness and answer relevancy exhibit significant variation across the models, suggesting that certain models, such as the Qwen variants and Phi 3.5 models, are better suited for tasks requiring accurate and consistent answers following the retrieved information and instructions given by the developer. In this respect, the surprising result of **Qwen2.5 0.5b** deserves a mention, which, being the smallest and thus theoretically the weakest model, scored highest in the answer relevancy metric.

Semantic similarity, on the other hand, shows relatively consistent performance across most models, indicating a shared ability to maintain semantic coherence in outputs.

When examining context precision, which measures how much of the information in the generated response is directly attributable to the retrieved context. Certain models, such as those in the Qwen and Phi series, demonstrate higher scores, indicating that their responses are tightly grounded in the provided evidence. This is critical for applications requiring reliability and adherence to the given context. Conversely, models with lower scores suggest

a tendency to include unsupported or external information, potentially leading to hallucinations or less accurate responses.

On the other hand, when analyzing the context recall scores, it is clear that the retrieval system often failed to find the correct information in the ground truth of the evaluation set. This indicates a gap in fully leveraging the available evidence to generate responses. The factual correctness scores are even lower, suggesting that, despite the retrieved information, the models often fail to produce factually correct responses in relation to the reference provided. This finding is strongly related to the low context recall scores.

These results, however, could also be influenced by how the reference was written and by the fact that the model cannot respond in exactly the same way as the reference. Moreover, the metric was calculated using the default parameters for atomicity and coverage³¹. However, by adjusting both of these parameters, it is possible to customize the level of detail and completeness for the specific case.

In Figure 5.2, we can see the same plot as before, but with metric values computed using `ChatGPT-4.0` as LLM evaluator instead of `LlaMA 3.1 70b` and `text-embedding-3-small` as embedding model instead of `m2-bert-80M-8k-retrieval`. Surprisingly, despite the significant differences between these two evaluation models in terms of training and capability, the results are strikingly similar. This unexpected consistency highlights the robustness and reliability of the RAGAs evaluation framework. One might have anticipated variations in the trends or relationships among the metrics due to differences in how each evaluator interprets the generated responses.

However, at least in our case, the tool has demonstrated its ability to produce stable, evaluator-agnostic results. This outcome reinforces confidence in RAGAs' capacity to provide accurate and consistent evaluations of model performance, regardless of the underlying evaluation model used.

³¹https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/factual_correctness/#atomicity

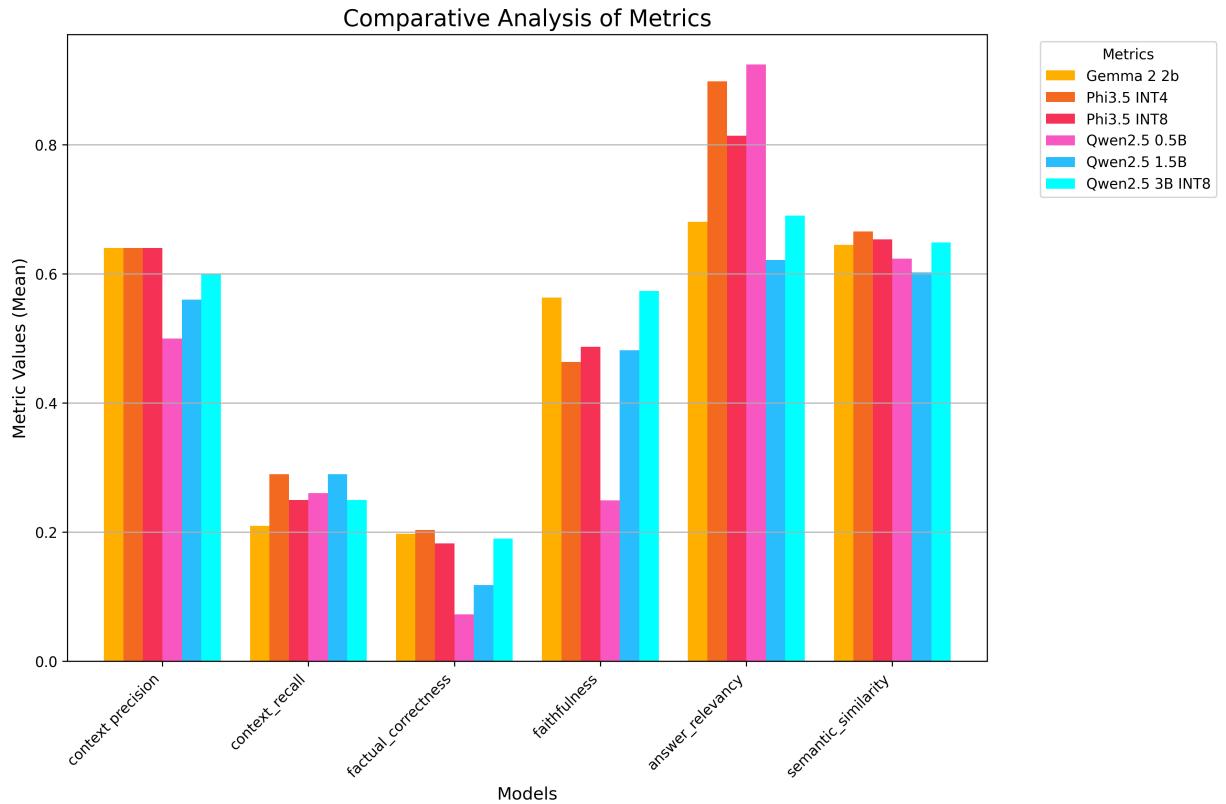


Figure 5.2: The metrics mean over all models computed using ChatGPT-4o

At this point we can analyze in the details the metrics for each user question and models.

5.5.2 Gemma 2 2b-it

We start with the **Gemma 2 2b-it** model. Results are in Figure 5.3. We see the heat map showing all the values of the metrics for each question represented by an index. We have used this visualization for all models to maintain consistency and because it allows, unlike line charts or bar charts, a clean visualization by not having values that overlap because they are very similar.

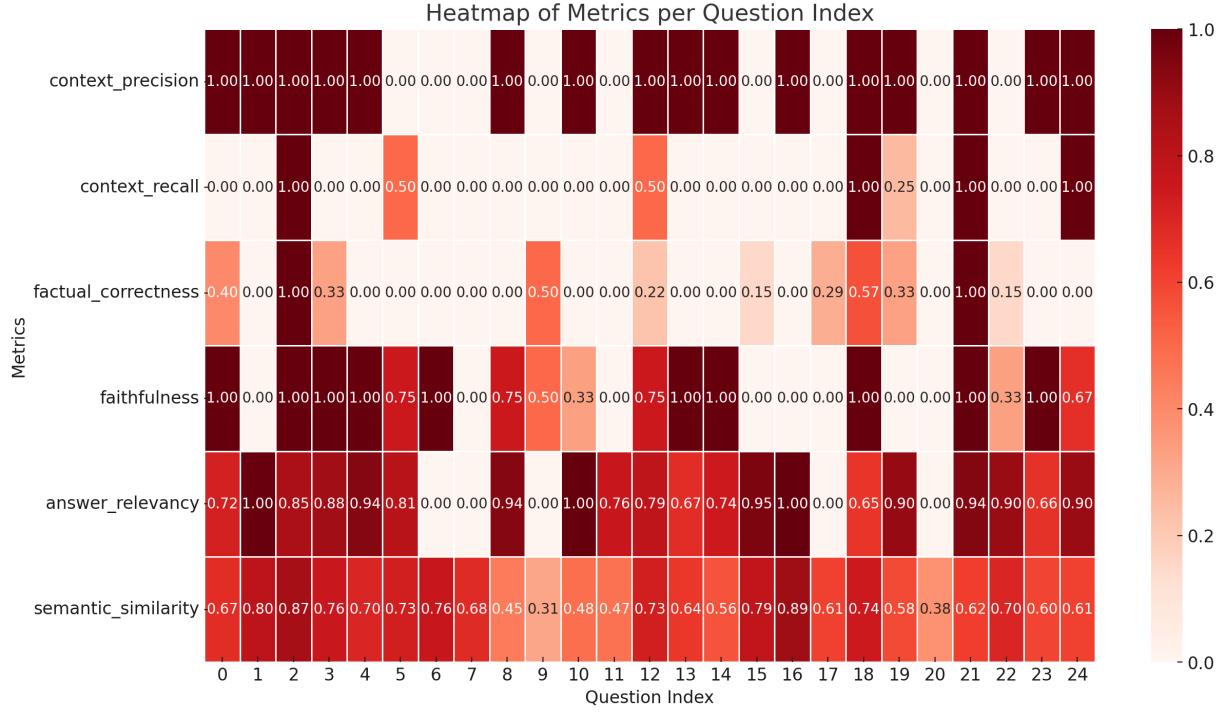


Figure 5.3: Gemma 2 2b metrics

The heat map for the **Gemma 2 2b**-it model offers valuable insights into the model’s performance across different metrics. The first observation concerns context precision, which is consistently high for most question indices, often reaching 1.0. This indicates that the model is highly effective at grounding its responses in the provided context and avoids introducing unsupported or hallucinated information. However, for indices with missing contexts (7, 9, 15, 17, 20), precision drops to 0.0, as expected, since the model cannot utilize non-existent evidence.

This indicates that the R.A.G. system manages to find relevant information most of the time. Sometimes, however, the R.A.G. cannot find information that exceeds the threshold score, so no context is generated.

When analyzing context recall, the variability across questions becomes apparent. In certain cases, such as indices 2, 9, 17, and 24, the recall reaches its maximum value of 1.0, demonstrating the ability of R.A.G system to generate a `retrieved_context`, which contains all the information in the reference. On the other hand, for indices where the RAG system failed to generate contexts (7, 9, 15, 17, 20), recall is unsurprisingly 0.0, as no relevant information was retrieved by the R.A.G.

Factual correctness, which evaluates the truthfulness of the model’s responses based on

external knowledge or independently verifiable facts, presents a more nuanced picture. This metric is not based, to be calculated, on the retrieved context, meaning that even for indices with empty contexts, the model can achieve non-zero scores. For example, while some indices with missing contexts (e.g., 7 and 17) score 0.0, others, such as 8, achieve reasonable correctness levels, reflecting the model’s ability to generate factually accurate answers when the required knowledge is already encoded in its parameters.

Faithfulness, on the other hand, remains high for most indices where contexts are available, frequently reaching 1.0. This demonstrates that the model often adheres closely to the provided evidence when formulating responses. However, as seen with other metrics, indices with missing contexts exhibit lower scores, as the absence of supporting evidence prevents the model from grounding its responses in the retrieved context.

The answer relevancy metric highlights the model’s capability to provide contextually appropriate answers. For most indices, the scores exceed 0.7 and often reach 1.0, showing that the model performs well in aligning its responses with the intent of the questions. However, at indices where the context is missing, such as 7 and 9, the scores drop significantly, sometimes to 0.0, reflecting the challenges posed by the lack of retrieved evidence.

Finally, semantic similarity, which measures the coherence of the response with the ground truth, shows moderate to strong performance across most indices, with values typically ranging from 0.6 to 0.8. Despite this, there are occasional dips, especially for indices with missing contexts, as the absence of grounding information negatively impacts the semantic alignment of the generated responses.

It is important to remember that if the answer and the reference are semantically similar, this is not a sufficient condition to know whether the answer is correct.

In summary, the **Gemma 2 2b-it** model demonstrates strong performance in context precision, faithfulness, semantic similarity and answer relevancy when contexts are available. However, its performance in factual correctness highlights its dependency on the quality and availability of retrieved contexts, as demonstrated by the results of the context recall. The empty contexts at indices 7, 9, 15, 17, and 20 reveal the critical role of the R.A.G system in enabling the model to fully leverage its potential. Despite these challenges, the model occasionally showcases its ability to generate accurate and relevant answers even in the absence of retrieved evidence. This variability underscores the need for optimizing both the retrieval system and the model’s ability to independently provide accurate, coherent responses.

5.5.3 Phi3.5 mini INT4

The heat map (Figure: 5.4) for the **Phi 3.5 mini INT4** model highlights the performance of this model across different metrics and question indices. A noticeable trend across the

heat map is the model's strong grounding in context precision, with high scores, often reaching 1.0, for most indices. This indicates that the model reliably avoids introducing unsupported information when the retrieved context is available. However, as expected, context precision scores drop to 0.0 for indices with missing contexts (7, 9, 15, 17, 20), reflecting the lack of any evidence to support its responses.

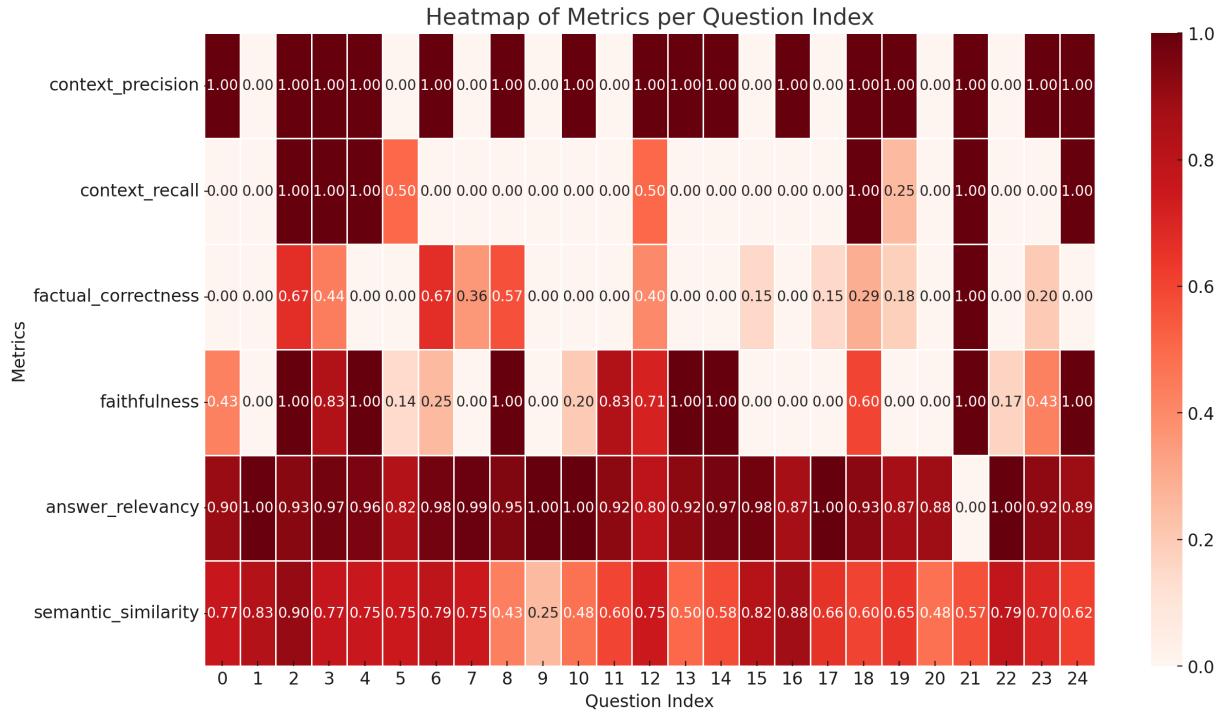


Figure 5.4: Phi 3.5 INT4 metrics

The context recall metric displays a more variable performance, with scores ranging from 0.0 to 1.0. The R.A.G. system is sometimes able to retrieve all the information in the reference (e.g. 2, 21 index) thus giving a big hand in the generation of the answers. Other times, excluding those where the retrieved context is empty, the system fails to find information relevant to the reference (10°question index).

Factual correctness presents an interesting distribution of scores across the indices. This metric evaluates the truthfulness of the model's responses by comparing them to the reference, but it is directly influenced by the retrieved context, as the retrieved evidence informs the content of the generated response. In some indices, such as 2 and 19, the model achieves high scores of 1.0, indicating that it produces factually accurate outputs in alignment with the reference. However, for indices with missing contexts (e.g., 7,

9, 15, 17, 20), the scores vary significantly. For instance, at index 9, the score drops to 0.0, reflecting the model's inability to generate factually correct responses in the absence of retrieved evidence to guide it. This variability underscores the interconnected nature of the retrieval process and the model's ability to produce factually accurate answers, highlighting that while factual correctness is assessed independently of the retrieved context, the retrieval step plays a critical role in shaping the responses.

The faithfulness metric remains relatively strong across indices with available contexts, with scores often exceeding 0.8 and reaching 1.0. This demonstrates that the model tends to adhere closely to the retrieved evidence when formulating responses. However, for indices with missing contexts, such as 7 and 9, faithfulness scores drop significantly, highlighting obviously the challenge of staying faithful to absent information.

Answer relevancy or response relevancy scores reveal the model's effectiveness in providing contextually appropriate answers to the question and prompt without digressing. For most indices, the scores are moderate to high, often surpassing 0.7. This highlights the importance of using instructional models.

Nonetheless, there are occasional drops for indices with missing contexts, such as 9 and 15, where the scores approach 0.0, suggesting that the absence of retrieved evidence negatively impacts the relevance of the model's outputs.

Finally, semantic similarity shows moderate performance overall, with values typically ranging between 0.6 and 0.8 for most indices. However, as with other metrics, missing contexts lead to noticeable dips, especially for indices 9, 15, and 17, where scores fall below 0.5. This reflects the impact of missing retrieval data on the semantic alignment of the responses.

In summary, Phi 3.5 mini INT4 model demonstrates strong performance in metrics like context precision and faithfulness when the R.A.G system successfully retrieves contexts. However, its dependence on retrieved evidence is evident in the sharp declines observed in context recall, faithfulness, and semantic similarity for indices with missing contexts. Interestingly, the variability in factual correctness across indices highlights that while the model can occasionally provide accurate, fact-based answers independently of retrieved information, it is not consistently reliable as the R.A.G system. This heat map underscores the importance of robust retrieval and evidential grounding to enhance the performance of the Phi 3.5 mini INT4.

5.5.4 Phi 3.5 mini INT8

The heat map for the Phi 3.5 mini model quantized to INT8 offers a granular view of the model's performance across various metrics and question indices. One notable observation is the strong performance in context precision, where the model frequently achieves a

perfect score of 1.0 across most indices. This indicates that the model reliably grounds its responses in the provided evidence when contexts are available. However, as expected, context precision scores drop to 0.0 for indices with missing contexts (7, 9, 15, 17, 20), as the absence of retrieved evidence directly affects this metric.

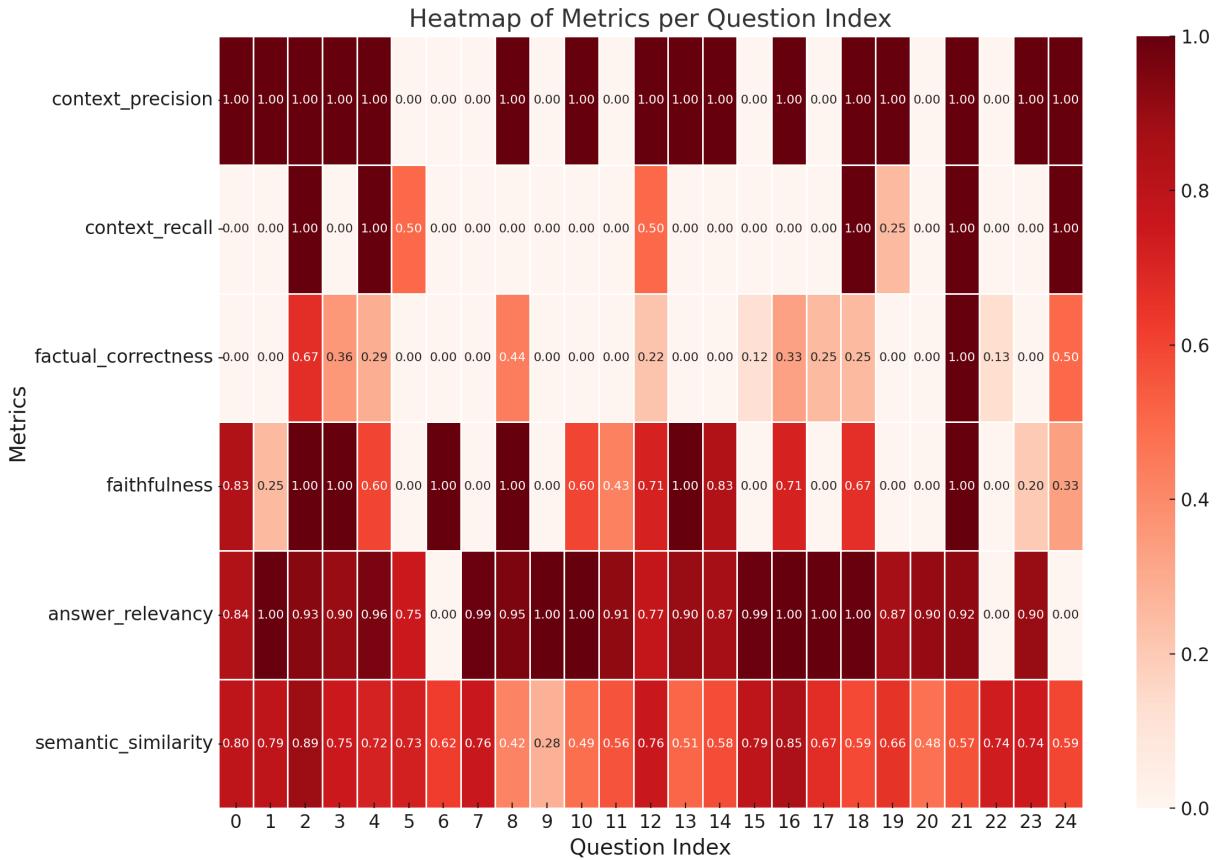


Figure 5.5: Phi 3.5 INT8 metrics

The context recall metric reflects the ability of the R.A.G. system to retrieve information relevant to the reference. In this case, the variability in scores highlights the performance of the retrieval system. For indices like 2 and 4, where recall reaches 1.0, the R.A.G. system successfully retrieves all the information that is also present in the reference. Conversely, for indices with missing contexts (7, 9, 15, 17, 20), recall is predictably 0.0, as the R.A.G. system was unable to retrieve any context. This underscores that the metric evaluates the retrieval step's success independently of the generation model's ability to use the context. The observed variability demonstrates that while the R.A.G. system can occasionally retrieve highly relevant information, there are cases where it completely fails, leading to gaps in the pipeline.

Factual correctness, while assessed by comparing the response with the external reference, is indirectly influenced by the retrieved context, as the generated response is shaped by the available evidence. In this heat map, the scores show a wide range. For instance, indices like 2 achieve high scores of 0.67, indicating factually accurate responses. However, for indices like 7 and 9, where contexts are missing, the scores drop to 0.0, reflecting the challenges in generating accurate outputs without evidence to guide the model. This demonstrates how the absence of retrieved contexts indirectly impacts factual correctness by limiting the model’s ability to generate factual responses.

Faithfulness scores generally indicate strong alignment between the model’s responses and the retrieved context. Many indices achieve scores close to 1.0, such as 2 and 3, suggesting the model effectively adheres to the evidence provided. However, at indices where contexts are unavailable or responses diverge from the evidence, such as indices 9 and 15, faithfulness scores decline, illustrating the importance of reliable context retrieval in maintaining fidelity.

Answer relevancy remains a strong metric for this model, with most scores exceeding 0.8. The model consistently produces contextually relevant answers for indices like 2, 4, and 12, where scores approach 1.0. However, indices with missing contexts, such as 9 and 20, show significant drops in relevancy, highlighting the model’s reliance on retrieved evidence to generate responses that align with the user’s intent.

Lastly, semantic similarity demonstrates moderate to strong performance, with values generally clustering around 0.7–0.8 for most indices. This reflects the model’s ability to maintain coherence between its responses and the ground truth. However, as seen in other metrics, indices with missing contexts experience noticeable drops, such as indices 7 and 15, where semantic similarity scores dip below 0.5, indicating diminished alignment when evidence is absent.

In synthesis, Phi 3.5 mini INT8 shows strong performance in context precision, faithfulness, and answer relevancy when contexts are available, demonstrating its ability to produce reliable, coherent, and contextually appropriate responses. However, the model’s performance is noticeably affected by the absence of retrieved contexts, as seen in the declines in context recall, faithfulness, and indirectly, factual correctness. These results underline the importance of robust retrieval in enabling the model to fully leverage its capabilities and consistently deliver high-quality outputs.

5.5.5 Qwen2.5 0.5b

In the heat map below (Figure: 5.6), it should be noted that there is one less demand as can be seen from the heat map index count. This is because at question 4³², the model simply did not generate any answers. Therefore, we did not calculate the metrics for that answer.

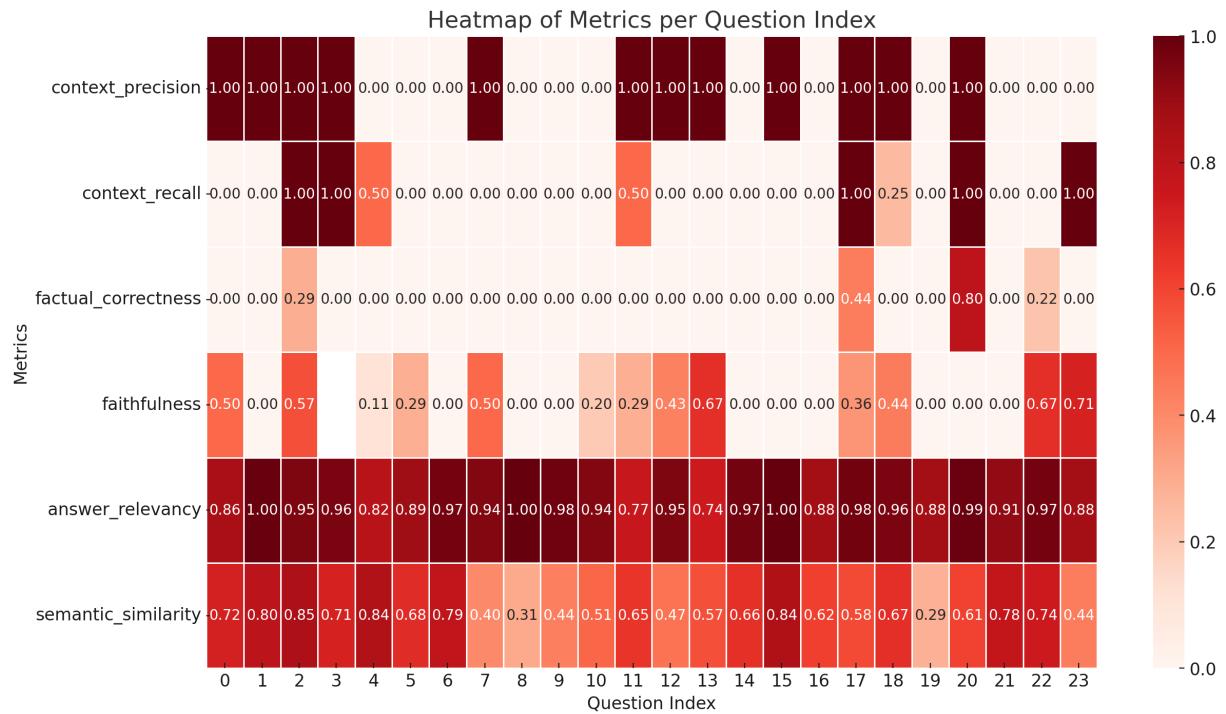


Figure 5.6: Qwen2.5 0.5b metrics

Despite its limited size, the model manages to demonstrate notable strengths in specific areas, although some significant challenges arise, particularly when context retrieval fails.

Regarding context accuracy, as with past models, **Qwen2.5 0.5b** manages to remain bound to the response reported in the retrieved context. For the context recall metric we can repeat what has already been said in previous comments for the same metric in the other models comments, since they refer only to the evaluation of the R.A.G. system, which, as we have already said, is identical even if you change the generation model. Therefore, this will also be the case for the next models.

³²At this link you can see this behavior before the response processing phase.

Factual correctness shows a large majority of values equal to 0 among the questions, as it depends on both the reference and the influence of the retrieved context on the model’s responses. At index 2, the model achieves a score of 0.29, indicating partial accuracy relative to the reference. In general, and taking into account the times when the `retrieved_context` is not available, we can say that the model, although supported by the R.A.G system, does not give factual answers. On the other hand, its size does not argue for different results.

Faithfulness scores highlight the model’s varying adherence to the retrieved evidence. For example, at index 2, the model achieves 0.57, indicating some degree of alignment with the retrieved content. However, for indices with missing or insufficient contexts, such as 4 and 15, the scores drop significantly, reflecting the difficulty for such a small model to maintain fidelity without a strong contextual base, and even when this is present³³.

Answer relevancy, on the other hand, remains one of **Qwen2.5 0.5b**’s stronger metrics. For indices like 1, 2, and 3, the scores approach 0.95, illustrating the model’s ability to produce contextually appropriate responses to user queries. Nonetheless, when the retrieval fails or contexts are missing, as at indices 7 and 15, relevancy scores decline, further emphasizing the reliance of small models like **Qwen2.5 0.5b** on robust retrieval to achieve strong results. Such high values for this metric with this model are really strange, especially in relation to those of other model³⁴, this behavior would deserves a more detailed analysis to verify that data are not affected by errors.

Semantic similarity is moderately strong across most indices, with values typically ranging between 0.7–0.8. This suggests that the model’s responses generally maintain coherence with the retrieved content and reference. However, the absence of retrieval at indices like 7 and 15 results in a noticeable decline in scores, reflecting the reduced ability to produce semantically aligned responses without sufficient contextual support.

In conclusion, while **Qwen2.5 0.5b** is a compact model, it demonstrates commendable capabilities in metrics such as answer relevancy and semantic similarity when supported by robust retrieval. However, its performance in terms of correctness and factual fidelity is significantly affected by the availability and quality of the retrieved contexts, as much as the model’s ability to utilize them when present with good quality.

5.5.6 **Qwen2.5 1.5b**

The heat-map for the **Qwen2.5 1.5b** model demonstrates its performance across all the question, reflecting its strengths and weaknesses as a small language model. While some trends align with observations from smaller models (**Qwen2.5 0.5b**), the **Qwen2.5 1.5b**

³³See the faithfulness for this model in Figure 5.2

³⁴See the comparison with the other models in Figure 5.2

also shows improvements in certain areas, particularly in its ability to handle more complex queries.

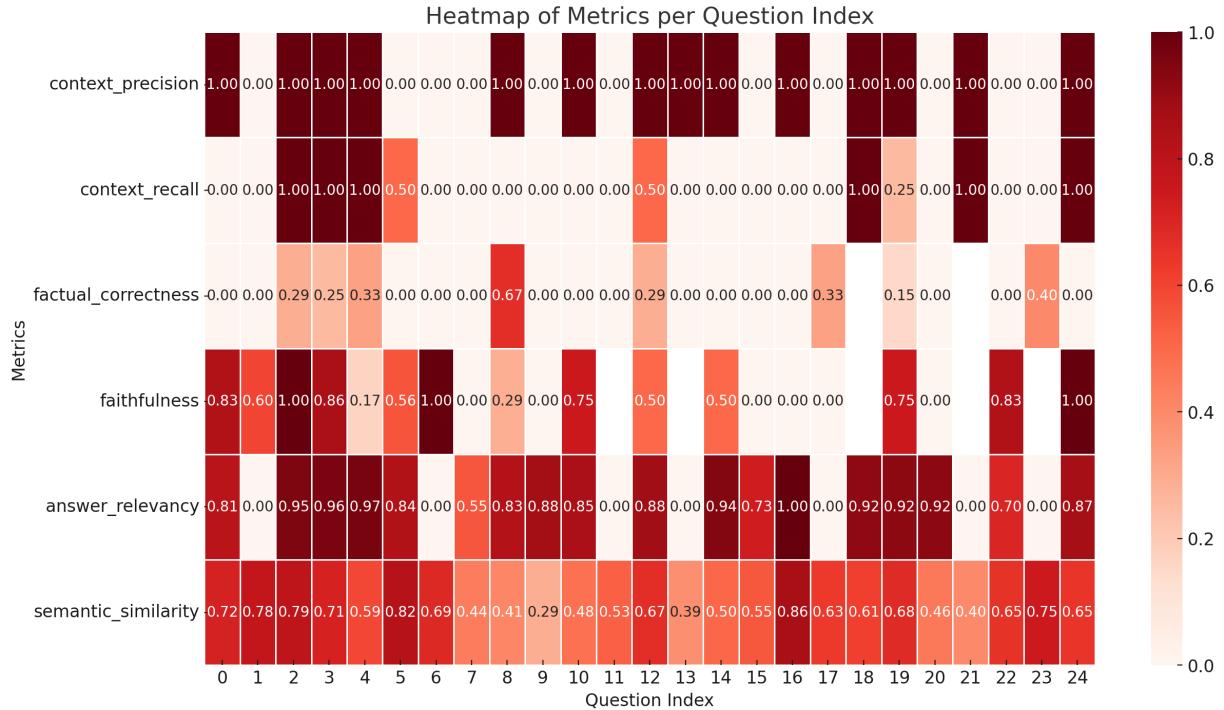


Figure 5.7: Qwen2.5 1.5b metrics

The context recall metric is similar to the values shown for the previous models, the changes are very minimal. This is probably due to how the re-ranker model assesses the importance of the information retrieved each time.

Factual correctness exhibits mixed results, highlighting the model's varying ability to produce accurate answers in alignment with the reference. For example, indices like 2 and 6 achieve moderate scores (0.29 and 0.67, respectively), indicating partial alignment with factual references. However, indices such as 7, 9, and 15 show scores of 0.0, demonstrating the challenges in generating accurate outputs when the R.A.G. system fails to retrieve relevant context or when the model cannot leverage existing evidence effectively.

Faithfulness scores show some variability, with notable highs such as 1.0 at indices like 2 and 6, where the model adheres closely to the retrieved context. However, there are instances, such as indices 3 and 9, where the score drops below 0.5, reflecting deviations from the retrieved evidence or insufficient alignment with the context. This suggests that while Qwen2.5 1.5b can effectively ground its responses when the evidence is strong, it

struggles in scenarios with incomplete or irrelevant contexts.

Answer relevancy remains a strength for **Qwen2.5 1.5b**, with scores frequently exceeding 0.8 for indices like 1, 2, and 13. These results indicate the model’s ability to provide responses that align well with the user’s queries. However, indices with missing or weak contexts, such as 7 and 15, show significant dips, reinforcing the importance of robust retrieval for maintaining relevance.

Semantic similarity demonstrates moderate performance overall, with scores generally ranging from 0.6 to 0.8. At indices such as 2 and 13, scores approach 0.86, showing strong alignment between the model’s outputs and the reference. However, for indices with missing contexts or incomplete grounding, such as 7 and 15, scores drop significantly, reflecting the model’s reduced coherence in these cases.

In summary, the **Qwen2.5 1.5b** model demonstrates notable improvements over smaller models, particularly in metrics such as faithfulness and answer relevancy, where it showcases the ability to generate more relevant and grounded responses. However, its performance in factual correctness and semantic similarity continues to be influenced by the availability and quality of retrieved contexts, as evidenced by significant drops in these metrics for indices with retrieval failures.

For this model, it is important to point out anomalous behavior. Often, instead of answering the user’s question or before doing so, the model generates, in some cases, the same question by repeating it in the answer or by asking a new similar question. This same behavior was also noted by my colleague Dr. Zanetta in his work, using the same model, as we shall see in the part concerning the GPU. This way of answering is probably due to the model’s inability to understand the template prompt, which we recall was not disclosed directly by the developers but extrapolated from the model files. However, this solution is not entirely convincing, since the other two models in the same family, which use the same prompt template, do not present this problem. Further tests will therefore have to be carried out in the future to understand what is happening.

5.5.7 Qwen2.5 3b INT8

This is the last model under consideration. This was the latest to be released for the Qwen2.5 series, but despite the remarkable capabilities highlighted by the developers, as well as its size, which made it one of the most balanced SLMs, unfortunately, as shown in Figure 5.8, the results were no better than those of other models in the Phi series.

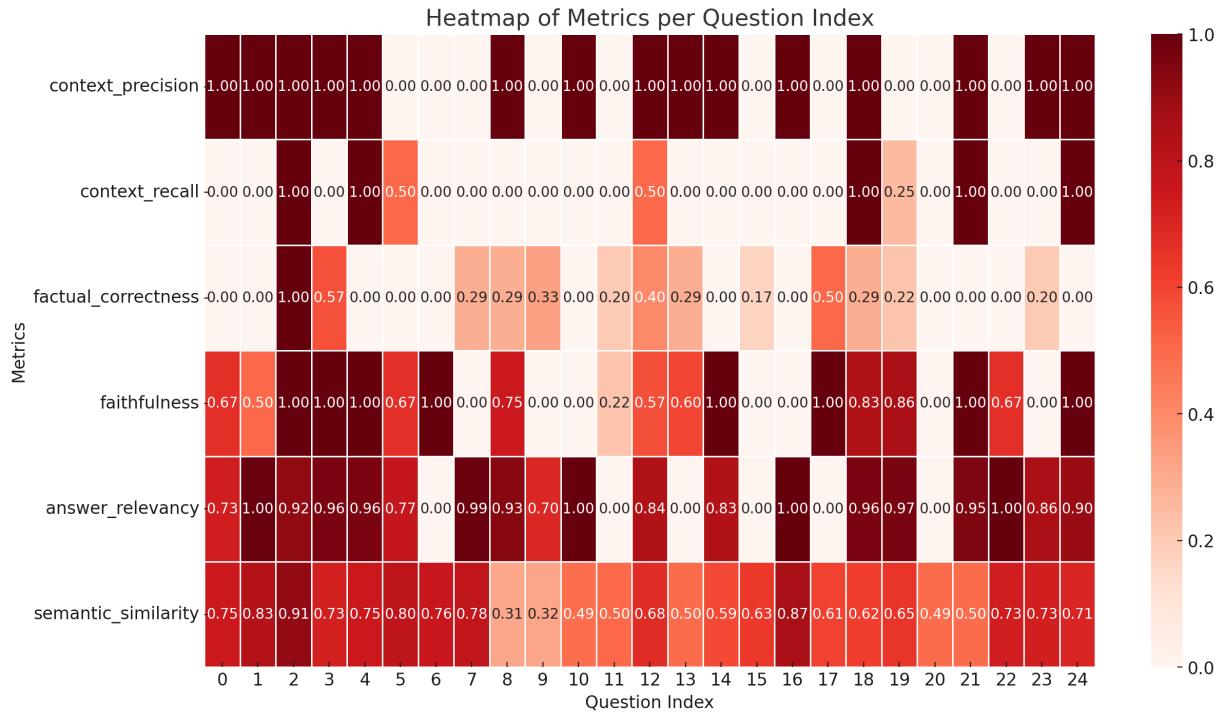


Figure 5.8: Qwen2.5 3b INT8 metrics

The context precision is approximately the same as for the other models. Thus, the context seems relevant, because the models are able to make use of the information it contains. The context recall struggles, again as with all other models, because the information retrieved often does not match the reference provided. However, this could also be due to the way the reference is written and then compared by the evaluating LLM.

Factual correctness is an area where the **Qwen 2.5 3B INT8** model demonstrates mixed results. For instance, at indices 2 and 3, scores of 1.0 and 0.57, respectively, indicate that the model can generate factually accurate responses when supported by strong retrieval. However, at indices 4, 9, and 16, factual correctness drops to 0.0, highlighting the limitations imposed by missing or irrelevant retrievals. This suggests that even a larger model like **Qwen 2.5 3B INT8** relies heavily on the quality of the retrieved evidence to produce factually accurate answers.

Faithfulness results reveal the model's ability to align its responses with the retrieved contexts. At indices like 2 and 6, scores reach 1.0, demonstrating the model's strong adherence to the evidence provided. Obviously this happens if the context exists for that question or if it is good enough. Which could also be a good thing but would have to be studied on a case-by-case basis.

Response relevancy remains one of the model’s strongest metrics, with scores often exceeding 0.9 for indices like 2, 4, and 16. These results indicate that **Qwen 2.5 3B INT8** consistently generates responses that are contextually appropriate and aligned with the queries. On the other hand, we have not relevant and appropriate answers to the original question for other questions such as 6 or 11.

Semantic similarity shows moderate performance, with scores generally clustering around 0.7–0.8 for most questions. This indicates that the model’s responses are reasonably coherent with the reference. At indices like 2 and 16, scores exceed 0.87, reflecting strong semantic alignment. However, indices such as 9 and 20 show noticeable drops, with scores below 0.5, illustrating reduced semantic coherence when retrievals are absent or weak.

5.5.8 Inference time analysis

In this section, we will examine the inference times for each model. It is important to note that extremely low inference times are not critical for our tool, as MarineChat is not designed for real-time use. A hypothetical user has the time to ask a question and wait for a response without needing an instant reply. Typically, navigation allows for taking the necessary time, except for emergency situations, where MarineChat would not be able to assist anyway.

In the Figure 5.9 we see the average times for each SLMs to get an overview and we can immediately state that all models have reasonable times to generate a response. None takes more than a minute on average.

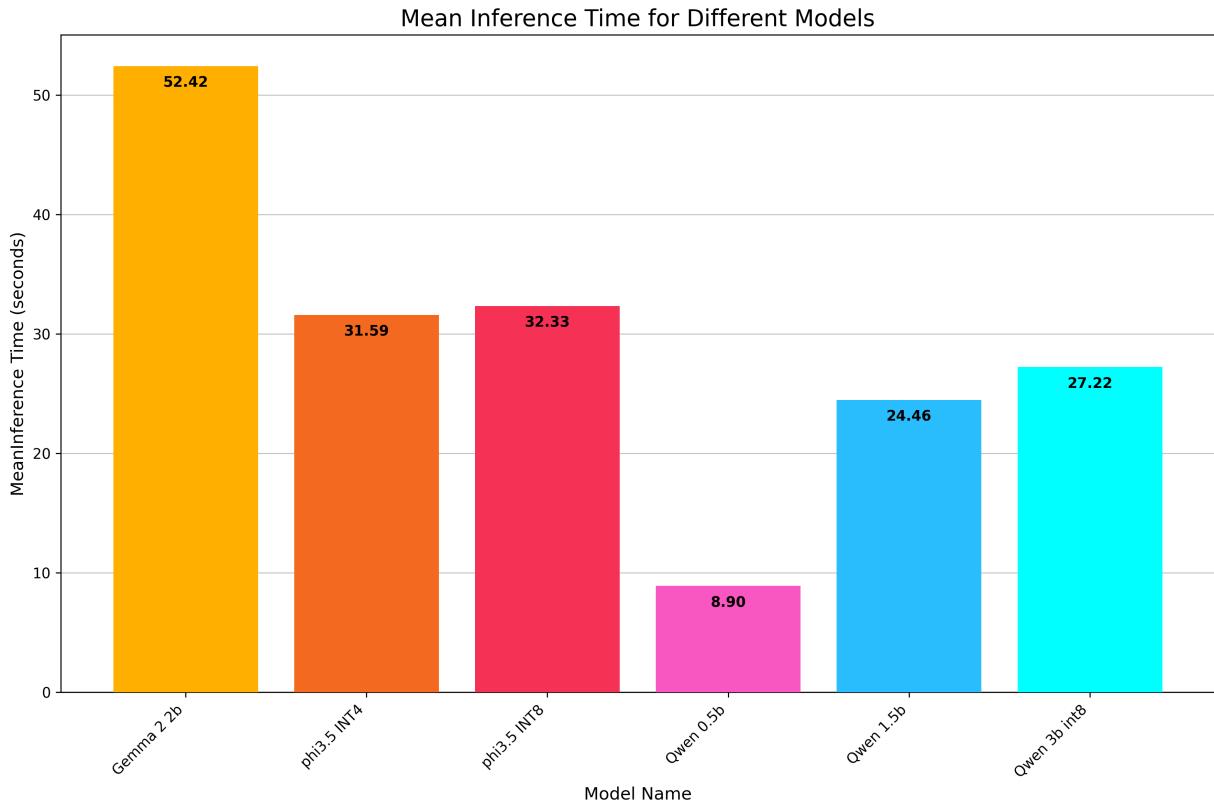


Figure 5.9: Average inference time for each SLMs

The Figure 5.10 displays the inference times taken by various models to answer the questions. The first thing that stands out is the inference time of **Gemma 2 2b-it**. It is evident that the inference time for this model is considerably longer than what would be expected for the initial question. This anomalous result is attributable to the fact that, prior to being answered, the model must be loaded from secondary memory, where it is saved. This is not the case for the other questions, as the process has already been completed. However, the subsequent questions are also slower than the other models, despite the fact that this is not the heaviest model. This is due to the version of OpenVINO used in this thesis (2024.3) not yet supporting the Gemma 2 models (2b & 9b)³⁵.

³⁵Release note of newer version of OpenVINO in which support for Gemma 2 b has been added



Figure 5.10: Comparison between the inference time for each questions for the different SLMs

Three models are very similar in terms of inference performance: Phi 3.5 mini INT8, Phi 3.5 mini INT4, and Qwen 2.5 3B INT8. Starting from comparable sizes (see the Table 5.1), their quantization results in similar inference times. Notably, the smallest model, Qwen 2.5 3B INT8, which has only 3.09 billions parameters (compared to Phi3.5's 3.82 billions), is also the fastest of the three.

Regarding these three models, it is interesting to note that despite being quantized to a less precise 4-bit representation, Phi 3.5 mini INT4 has an inference time almost identical to its larger counterpart, Phi 3.5 mini INT8. This is because the quantization was performed asymmetrically, with a ratio that retains 20% of the weights in INT8 precision. Consequently, at least in terms of inference performance (though the INT4 model is smaller in memory, taking up only 3 GB compared to the 5 GB of the INT8 version), there is virtually no difference between INT8 and INT4_ASYM.

Regarding the other two models, Qwen2.5 1.5b shows intermediate performance compared to Qwen2.5 0.5b and Qwen2.5 3b INT8. Its inference times are slightly slower

than `Qwen2.5 0.5b` but faster than `Qwen2.5 3b INT8`, making it a balanced option for those requiring moderate model capacity without the resource demands of larger models. `Qwen2.5 0.5b`, on the other hand, is the fastest overall, with consistently low inference times. This is expected due to its much smaller size (0.5 billions parameters), making it an ideal choice for scenarios where speed and computational efficiency are critical, even if some complexity or capability is sacrificed.

5.6 Test on GPU

Let us now look at the tests carried out on the GPU. These are mainly done by BSc Stefania Zanetta in her thesis. Here we will only see a general overview of how good the results of the evaluation metrics are, and as before in the CPU part we will see the inference times for each question using the different models.

The results were obtained by passing to BSc Zanetta the index vector created using the dataset for our use-case and the set of queries to generate the answers. These were generated, using the same models, except for `Gemma 2 2b-it`, for reasons related to the impossibility of using updated libraries on the Jetson Xavier required for the inference of that model and `Qwen2.5 1.5b`, because BSc Zanetta encountered a strange behavior, for which she has not yet found a reason and a solution, whereby the model answers with a different or the same question posed by the user. Furthermore, the models were not quantized because, using a GPU, BSc Zanetta did not find this optimization useful.

All the parts of the pipeline that form the MarineChat, have been maintained also in the GPU apart from OpenVINO-related functions and classes. The model for the embedding step and the model for re-ranking are also the same, allowing for the clearest possible comparison of results.

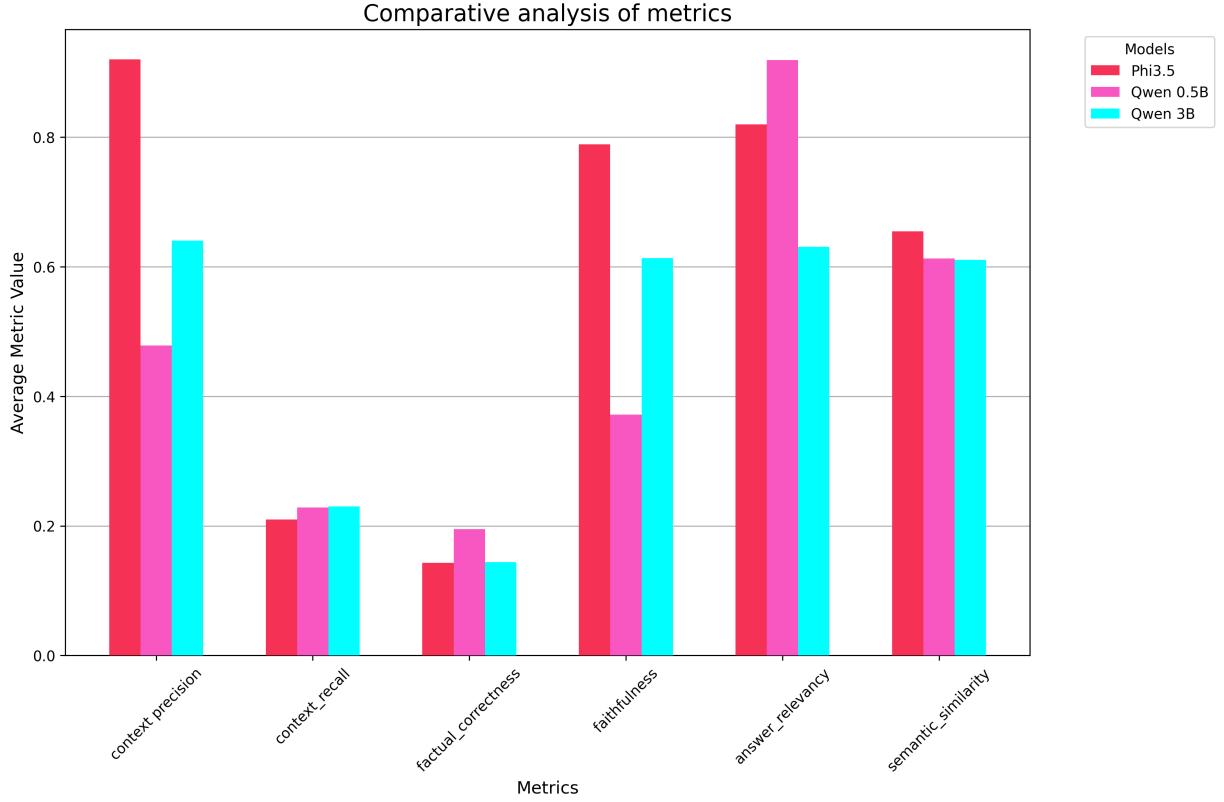


Figure 5.11: Mean of the metrics using as LLM evaluator ChatGPT-4o

The Figure 5.11, show the metrics computed using the OpenAI models as evaluators. We note that **Phi3.5** model performs best in many metrics, such as context precision, which indicates that the retrieved context is used in the answer given by the model, or faithfulness, which again highlights, using another formula, the model’s ability to take into account the information found. The best results of this model on the GPU are achieved by using the model without quantization.

Qwen2.5 0.5b is the smallest model of all and yet it performs well overall. It even beats the larger models, especially in factual correctness and answer relevancy. Improved results on CPU (Figure: 5.2). This is due to the non-determinism of the language models’ answers, which makes them more difficult to evaluate, especially in the absence of very large evaluation sets [SWLL24]. On the other hand, the last model, **Qwen2.5 3b**, performs particularly well in context recall, the only metric in which it outperforms its two competitors, but this result is not attributable to the SLM but to the part of R.A.G that worked slightly better by showing that small part of the system’s nondeterminism related to the re-ranking model.

In this part, only a quick analysis of the results was made for my use-case using a GPU device, as our work was focused on CPU usage.

5.6.1 Inference time analysis

Let us look at the inference times for the three models. As already mentioned, very low inference times are not necessary for our use-case, even on GPUs, but it is still interesting to see these times.

The Figure 5.12 shows the average inference times for each model. As expected, **Qwen2.5 0.5b** is much faster due to the lower computational load. The other two models are very similar, 28.98 seconds and 28.43 seconds, in terms of average times, with **Qwen2.5 3b** being slightly faster, reflecting the fact that **Qwen2.5 3b** has slightly fewer parameters (3.09 billions of parameters instead of 3.82).

Compared to the CPU times (In Figure: 5.9), **Phi3.5 mini** un-quantized generates the text in less time on average. There is a slight worsening with the GPU for **Qwen2.5 0.5b** probably due to different amounts of text generated the same probably due to the slight difference for the worse for **Qwen2.5 3b**.

We could have expected much better results with the GPU, but we have to take into account that the device used by BSc Zanetta is older and also did not use any libraries, techniques or frameworks to speed up the inference, unlike us with OpenVINO, mainly due to compatibility problems of more modern tools with the hardware.

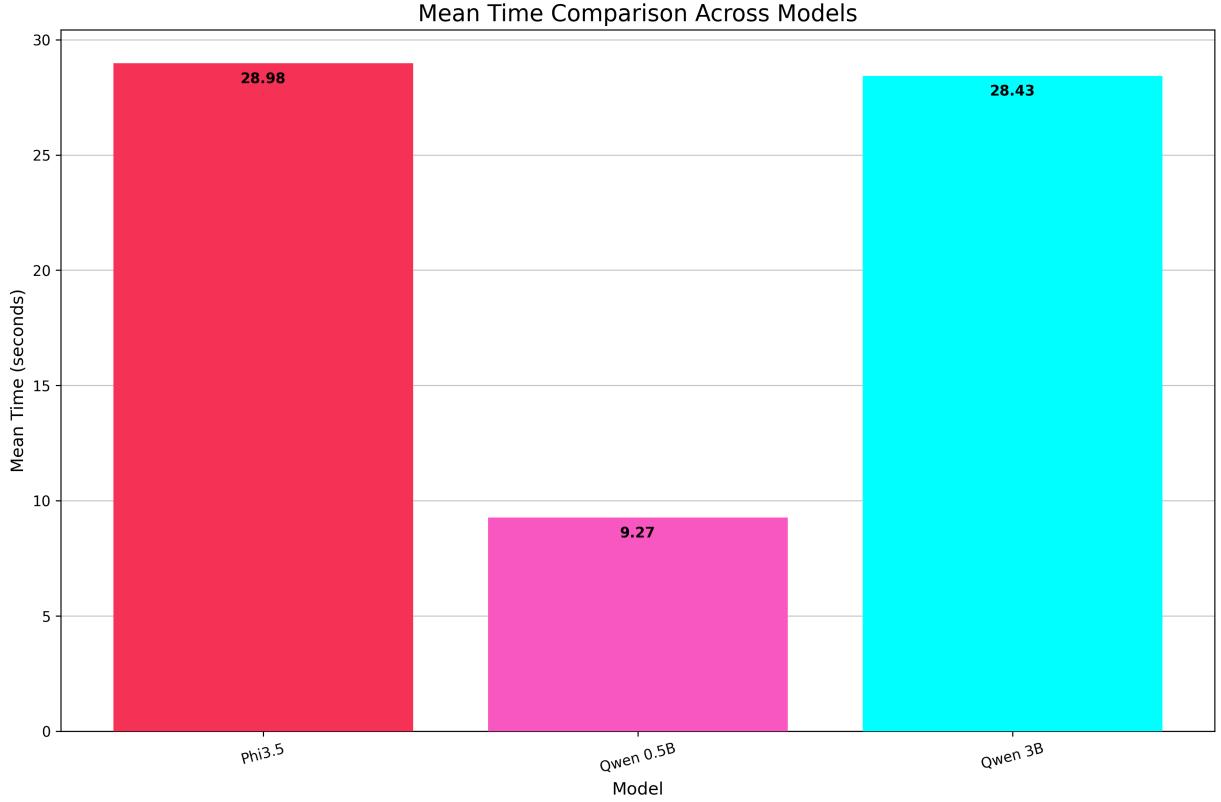


Figure 5.12: Mean inference time for each model on GPU

5.7 Discussion

The results obtained with regard to the qualitative evaluation of the model's answers show that all the models tested are able to remain faithful to the user's question and to the information retrieved to answer them in the majority of cases. As can be seen from the average values of the metrics such as context precision, response relevancy, faithfulness and semantic similarity. Given the size of these models, this outcome was not guaranteed and allows us to consider the SLMs as suitable for integration for final generation in R.A.G system on edge device and in no-Internet connection environments.

On the other hand, as repeatedly emphasized in the comments on the results for the various SLMs, the results show that the context retrieved by the retrieval part of the R.A.G pipeline are often not useful or, insert irrelevant information which the model then use to answer the question posed by the user in a totally or partially incorrect manner. This issue is reflected in the generally low values of the context recall metric, which, as a

reminder, measures whether the information from the reference is present in the context and thus indicates whether the context can help the model provide the correct answer or not. Similarly, it is reflected in the factual correctness metric, which, aims to determine if the generated response aligns with the provided reference and generally shows very low values.

The inference times of the models are acceptable on both CPU and GPU devices. We would have expected lower times on the GPU [LXH⁺24] for every model, but this is only true for `Phi3.5 mini`, which, however, is not quantized. This suggest that if it were quantized, it would likely be even faster than the `Phi3.5 mini INT8` and `Phi3.5 mini INT4` models on the CPU.

The other two models tested in both devices (`Qwen2.5 0.5b` and `Qwen2.5 3b`) are only slightly slower, for the the first one the difference is probably due to the number of test and the varying amount of input as prompt passed to the model or the length of the generated output text. For the second model, we should remember that in the CPU the `Qwen2.5 3b` model was quantized to a INT8 precision so it is slightly faster because is smaller. In practice, the GPU is better as we would expect, but using optimization tools such as OpenVINO and compression techniques like quantization, along with more modern CPUs, can yield inference times similar to those with older GPUs. This demonstrates that the common belief that using a GPU for language model inference is better than using a CPU is true, but if certain precautions are taken, CPUs can also be utilized effectively for language model inference.

Chapter 6

Conclusion and future directions

In this thesis, we have presented a quantitative and qualitative evaluation of the integration of language models on edge devices for specific topics, capable of revealing the limitations of these models; consequently, a possible method to mitigate these limitations. To achieve this, we looked at the use of SLMs, which guarantee inference on edge devices; as an innovative technique to improve the responses generated, given the particular context of our use case, where SLMs struggle due to their intrinsic limitations, we wanted to test R.A.G.. The result is MarineChat, our prototype chat-bot R.A.G. designed to be used by sailors on small/medium sized boats to get port information for their needs, specifically optimized for use on edge devices based on CPUs.

Through the thesis, we first described what LLMs and SLMs are, then we introduced two essential techniques for their use: instruction tuning and model compression, followed by the definition of two fundamental elements for our tool: chat-bots and R.A.G.¹.

In the third Chapter, we examined the state of the art of three fundamental elements for the realization of the thesis: the most performant edge devices, the most capable models and the most useful tools/frameworks.

In the fourth Chapter, we described the implementation of MarineChat in its most important parts. In Chapter 5, we showed the results of our experiments on CPU and GPU for the different SLMs chosen, both in terms of quality and execution time.

Regarding execution times, we observed that the optimizations employed to speed up the inference process on CPU produced times comparable to those obtained via GPU.

Testing chat-bots based on SLMs with a R.A.G. of this kind was a challenge in itself, but by using the RAGAs framework, we were able to achieve the desired results. These tests, in our opinion, indicate, on the one hand, that our choice was the right one: in fact, the SLMs demonstrated their ability to answer questions on particular topics using R.A.G, following the given instructions, thanks to instruction tuning. Indeed, if one take a look

¹See Chapter 2

at the cases where the retrieved context was empty (indexes of questions: 7, 9, 15, 17, 20 see for instance Figure 5.5) will see that the SLM are not able to respond in a correct manner based only on its knowledge. On the other hand, the same outcomes indicate that this latter aspect, the R.A.G part of the tool, needs to be improved to ensure the most accurate answers possible.

In our opinion, the direction for future improvements should first of all, concentrate on what has just been noted. There are many possible improvements for the R.A.G pipeline (see Figure 6.1) and new ones are being presented all the time [JZH⁺24], [ZZY⁺24]. The first horizons to be explored could be improving the clarity of the dataset using post-processing techniques, such as the extraction of images and tables ², the search for improvement of the pipeline testing different values for parameters such as chunk size (chunk optimization) or the `threshold_score`, or testing different embeddings [CDZ⁺24] and re-ranking ³ models maybe, first of all trying a non-quantized re-ranker ⁴. As far as the generators part is concerned, an interesting future development could be to broaden the look towards new SLMs ⁵ [HTH⁺24], new quantization techniques [WZS⁺24] and new libraries for inference ⁶.

A future working perspective, including the tool as a single entity, could be the use of multi-modal models. Those are SLM-based models with the ability to receive, reason and produce multi-modal information in various formats such as images, audio, video [HOC⁺24], [AJA⁺24], [LLLL24], [YYZ⁺24]. In addition, new methods of R.A.G could be explored, such as hybrid R.A.G [SMH⁺24], adaptive retrieval such as self-rag [AWW⁺23] or R.A.G designed to exploit different types of data formats, for example images [JGKS24] or maps.

Other future works could include: add the possibility of interfacing with MarineChat by voice and try to integrate it into on-board boat devices, such as modern chart-plotters. Finally, one might think of increasing the relevance of the evaluation set, by improving it through the use of experts in the field, enlarging it with the use of synthetic data generation techniques ⁷, or try new evaluation tools [SFKPZ23], carry out more in-depth tests on the results to better understand the problems.

²<https://docs.unstructured.io/welcome>

³<https://huggingface.co/jinaai/jina-reranker-v2-base-multilingual>

⁴Recall, that to improve execution times we quantized the re-ranker, but given the times obtained we could try not quantizing it, losing in speed but gaining in performance.

⁵<https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/>

⁶<https://llamaedge.com/>

⁷https://docs.ragas.io/en/stable/concepts/test_data_generation/rag/

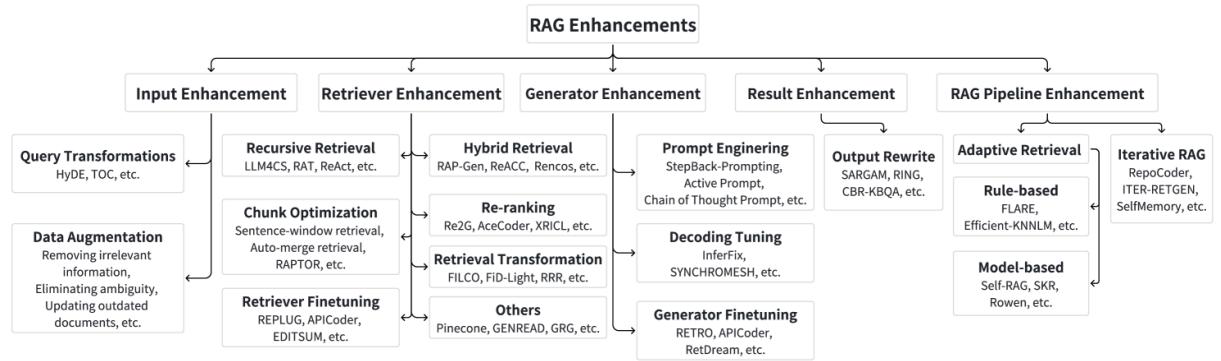


Figure 6.1: A picture of possibles R.A.G enhancements

Appendix A

Dataset

Table A.1: The question and ground truth

Question	Ground Truths
What are the typical winds in Genoa?	The typical winds in Genoa are in winter from N (tramontana) to NW, while in summer from SE (Scirocco) and SW (Libeccio).
What is the depth of the sea in the port of Alassio?	The average depth in the port of Alassio is 4m.
What is the VHF channel of the port of Imperia?	There are two VHF channels in the port of Imperia: 12 and 16.
Where is situated the Golfo Marconi?	Marconi Gulf is located between Portofino and Chiavari, along the eastern Ligurian Riviera.
Can you tell me the email address of the port of Ischia?	Ischia Harbor Email is ischia@guardiacostiera.it .
Can you illustrate how one should enter the port of Genoa, what instructions should I follow and what should I watch out for?	To enter the port of Genoa, one must have a moderate speed and be careful to use the right entrances for pleasure boats and control the movement of larger ships of all types as it is a very large port. If possible, contact the port offices first.

Question	Ground Truths
What is the intensity of the tides in the port of Ancona?	Tides in the port of Ancona are generally negligible, only reaching 0.5 m and up to a maximum of 0.9 m when the strong N wind (bora) blows.
What are the main dangers in the port of Naples?	The main dangers in the Port of Naples are the many fast-sailing tourist ferries.
What winds are sheltered from in the harbor of Otranto?	To all except N winds, which can be very violent, especially in winter.
Is it possible to land on the island of Pianosa in the Tremiti Islands?	No, because it is a marine protected area. One must keep a distance of 500 m from its coastline.
What are the prominent landmarks to identify Capo Mele from the sea?	A light is shown from a prominent tower with a dwelling, 25 meters high, standing on the cape. A signal station surmounts the summit of the cape.
Where can I anchor near the port of San Remo?	Vessels can anchor, in a depth of 12 meters, sand, close east of the harbor entrance. The holding ground is good, but anchorage is not advisable with strong south (S) winds.
Please can you describe the port of Carloforte?	Carloforte has a small harbor protected by two breakwaters, one almost perpendicular to the other, forming a narrow entrance.
Is there a VHF channel to contact for information about the passage of the Strait of Messina?	The VHF channels are 10, 13, 14, and 16.
To what winds is the port of Trapani exposed?	The harbor is exposed to all winds and is open to the W. In the spring, strong seasonal SE winds can make entry hazardous.
What is the marrobbio phenomenon?	Marrobbio is a rare meteorological phenomenon observed in the Mediterranean, particularly along the southwestern coast of Sicily. It involves sudden and significant oscillations in sea level, similar to a small tsunami, caused by atmospheric pressure changes and wind patterns. These rapid changes can lead to temporary flooding in coastal areas.

Question	Ground Truths
What is the maximum vessel size that can be accommodated at the port of San Remo?	Vessels up to 100 meters in length and with a draft of 4.3 meters can be accommodated at the port of San Remo.
What are the navigational hazards near the island of Giglio in the Tuscan Archipelago?	A restricted area lies 4 miles W of Isola Giglio. Mariners should consult local authorities before entering the area.
What are the possible dangers in entering the port of Civitavecchia at night?	Care should be exercised when entering the harbor at night as the navigation lights are not easily distinguished from the lights of the town.
Can you describe to me the typical winds that affect the port of Trieste in winter?	From September to March, winds alternate usually between the NW and NE. The NE winds sometimes blow with such force as to cause a sea in Mare Piccolo. Winds from the SW raise a lively sea and cause high water in Mare Grande. Winds from the SE, especially in winter, are strong and may disturb cargo handling by vessels berthed in Porto Mercantile.
What currents are there in the port of Chioggia?	Currents are strong and are mostly influenced by rain, runoff, and winds. With strong winds from the NE and SE, the current between the breakwaters can attain a rate up to 4 knots.
What is the draught depth in the port of Rimini?	The harbor has depths of 2.9 to 3.7m and is subject to silting.
Is the roadstead in Portoferraio Bay safe for anchorage?	The roadstead of Portoferraio affords anchorage to vessels of all sizes, but the bottom, being of soft mud, is not a good holding ground. With SE winds, which prevail in the area, vessels are liable to drag anchor, even when riding to two anchors. It is important to stay clear of the ferry turning areas and the harbor approaches. While ferry wash can sometimes cause some movement, it does not pose a significant danger for anchored vessels.

Appendix B

Document Excerpt

ID	Information Required
A	Vessel's type, name, call sign and flag (only if foreign)
B	LT of finished with engines
C	Pilot's name
D	Vessel position
E	Possible messages, should the pilot learn of the existence of serious faults which might put at risk the safety of shipping in general or the vessel concerned.

Vessels carrying dangerous and/or polluting cargo bound for Livorno must complete a check list and make it available to the Pilot on boarding. It is the duty of the pilots involved in the operations of approaching, leaving, mooring, and maneuvering of vessels to immediately inform the Port Authorities of Livorno of any serious deficiencies which might put at risk the safety of shipping in general or the vessel concerned.

Regulations.—A traffic separation scheme (TSS) with inner and outer precautionary areas, the limits of which are shown on the chart, lies in the approaches to the S entrance of the harbor and extends SW. Stopping is prohibited in the outer precautionary area. Inbound vessels must keep to the S of this zone and outbound vessels must keep to the N of it. Crossing the separation zone from N to S or vice-versa is prohibited except for



Livorno—Torre del Marzocco

small vessels and fishing boats.

If the entrance is occupied by a vessel leaving, an arriving vessel must wait until the entrance is clear.

Vessels using the N entrance must keep to the starboard side of the entrance channel. Vessels with drafts of more than 3m

Figure B.1: Example of an image in the document

water obstructions have been established 1 mile SW of Arenzano. Vessels must remain more than 80m distant from position 44°23.3'N, 8°40.8'E.

Voltri (44°26'N., 8°45'E.) is situated on the E side of Torrente Cerusa, 3 miles NE of Arenzano. It forms the W end of the built up area which extends W from Genova. A prominent belfry, with a cupola, stands near the shore between Torrente Cerusa and Torrente Leiro, which divides the town in half. A prominent church, with a white and red belfry, stands on a hill, 159m high, behind the town.

Torrente Cerusa and Torrente Leiro bring down considerable deposits which form temporary banks off their mouths. These banks are usually dispersed along the coast during heavy weather and cause the beaches to extend further seaward.

Pra (44°26'N., 8°47'E.) is situated 1.5 miles E of Voltri. This town may be recognized by a distinctive building standing on the heights behind it. The building consists of two blocks, the higher one being surmounted by a cylindrical tower.

Genova (Genoa) (44°24'N., 8°56'E.)

World Port Index No. 39470

10.17 Genova, the principal commercial port of Italy, lies centered 19 miles NE of Savona and also has extensive ship-repair facilities. The port handles containers, raw materials, bulk cargoes, chemicals, oil, perishable goods, and manufactured products. It is also a major ferry port and cruise line destination.

Port of Genova Home Page

<http://portofgenoa.com>

Winds—Weather.—The prevailing winds in summer are from the SE (Scirocco) and the strongest winds are from the SW (Libeccio). The latter seldom reach the port with any great force. However, when they do, these winds are quite violent and cause a high swell in the harbor.

Tidal Ranges for Genova

HAT	0.3m
MHWS	0.3m
MHWN	0.2m
MSL	0.15m
MLWN	0.1m
MLWS	0.0m
LAT	0.0m
Note —Predicted heights are in meters above the charted datum.	

Winds from the N to NW, especially in winter, descend from the mountains in violent squalls and frequently bring rain, sleet, or snow. They usually reach their maximum force at a distance of 1 to 2 miles offshore where they raise heavy seas.

Gales are most frequent in February and least frequent in July.

Tides—Currents.—See the table titled **Tidal Ranges for Genova**.

The tides are negligible with a rise of only about 0.3m at springs. However, changes in the water level caused by winds and atmospheric pressure variations can exceed those of the tide. Winds from the S tend to raise the level and winds from the N tend to lower the level.

Tidal currents within the harbor are negligible. Outside the harbor, the current generally sets W at an average rate of 0.7 knot, increasing with SE winds to about 2 knots.

Depths—Limitations.—The Multedo Offshore Oil Terminal is situated off the W end of the port and consists of a platform and a single buoy mooring.

The single mooring platform lies in a depth of 50m, 1 mile SW of the W end of the airport at Genova. A submarine pipeline extends NE from the platform towards the shore. The berth at this platform can handle tankers up to 500,000 dwt.

The single buoy mooring (SBM) lies in a depth of 65m, about 1 mile ESE of the platform. A submarine pipeline extends NW from this buoy towards the entrance to the inner oil harbor. The berth at this buoy can handle tankers up to 270,000 dwt, 350m in length, 53m beam, and 20m draft.

The W part of the main port consists of Porto Petrolì di Multedo, an inner oil terminal, the Ansaldo Shipyards, and the Cristoforo Colombo Airport. A steel works, with storage areas, is situated close E of the airport.

The E part of the port consists of numerous basins with the central part of Genova arranged like an amphitheater on the slopes of the mountains which encircle it.

The main harbor is protected from the S by a detached breakwater nearly 3 miles long. It is known as Diga Foranea, in the W part, and Molo Duca di Galliera, in the E part.

The entrance channel and turning basin at Port Petrolì di Multedo have depths of 15.5m. This inner oil terminal has four piers projecting from the N side and a total of eleven berths. Tankers up to 130,000 dwt, 259m in length, and 14m draft can be accommodated.

The W harbor entrance, which leads S of the W extremity of Diga Foranea, has a least charted depth of 13.5m. Use of this entrance is usually restricted to vessels berthing at the steel works basin. This basin, which has 500m of berthing space depths of 12.1 to 13.4m alongside, is used only by vessels with cargoes of coal or iron for the steel works. Vessels with a mast-head height exceeding 31m are not permitted to berth at the steel works.

The E and main entrance to the harbor lies between the E extremity of Molo Duca di Galliera and the head of Molo di Sottolutto, which extends SSW.

Avamporto is entered from the E entrance and comprises the area between Molo Duca di Galliera and Molo Giano. This basin is mostly used by vessels awaiting a berth.

Bacino delle Grazie lies N of Avamporto and is used mainly for repairs. Calata Olii Minerali, at the S part of the W side of this basin, has 474m of berthing space and can accommodate vessels up to 11m draft.

Porto Vecchio is entered from Bacino delle Grazie and extends W and N to the head of the port. There are depths of 7 to 11m alongside the numerous piers and quays which have facilities for general cargo, ro-ro, automobile ferries, and passenger

Figure B.2: Excerpt from the World Port Index relevant to Italian ports

Torre d'Ofanto, prominent and white, rises above the trees, 0.7 mile S of the river mouth.

17.19 Margherita di Savoia ($41^{\circ}23'N$., $16^{\circ}08'E$.), a small town, appears as a line of light-colored buildings rising from the sea. Two short moles extend N from the shore fronting the town and protect the entrance to a canal which is used by small craft. Three steel bridges span the canal and are prominent. The harbor authority monitors VHF channel 16. Anchorage can be obtained, in good weather, in depths of 11 to 13m, mud, about 1 mile off the canal entrance.

Caution.—A fish farm, marked by lighted and unlighted buoys, is located 2.5 miles NE of the town.

A dangerous wreck is reported to lie about 3 miles NE of the canal entrance.

The conspicuous village of Trinitapoli, with a prominent cupola, is situated 3 miles W of Margherita di Savoia and the more elevated village of San Ferdinando stands 6 miles SW of the town.

The village of Zapponeta stands 9 miles NW of Margherita di Savoia; its houses, with some red-roofed sheds, appear prominently near the beach.

This entire stretch of coast is recognizable by the conspicuous aerial cableway pylons which carry salt from Margherita di Savoia to Barletta.

Manfredonia ($41^{\circ}37'N$., $15^{\circ}55'E$.)

World Port Index No. 40590

17.20 Manfredonia, a small port, is divided into two harbors. The old harbor is formed by two converging breakwaters. The industrial harbor is situated at the head of a breakwater jetty which extends 1.2 miles SE from the shore at the NE side of the old harbor; piers at the head form a sheltered harbor basin.

Winds—Weather.—Winds from the SW sometimes blow briefly in heavy squalls and make entering and leaving dangerous. Although the roadstead offers little protection from sirocco winds, which blow from the SE and S, these seldom blow hard near the land and the sea is never very heavy in the roadstead. The bora, locally called monterese, which blows from the NE and N, sometimes arrives in violent squalls flowing down the gorges of Promontorio del Gargano.

Depths—Limitations.—The old harbor has 700m of total berthing space with depths of 2 to 7.5m alongside. It is used by small general cargo vessels, passenger ferries, fishing vessels, and naval craft with drafts up to 9.5m.

The industrial harbor, which has 5 berths, is mainly used for chemicals and bulk cargoes. It has 1,765m of total berthing space, with a depth of 10m alongside, and can accommodate vessels up to 9.9m draft with no restriction in length.

Aspect.—Monte Calvo, 1,055m high, stands 9 miles NW of Manfredonia and is the highest and most prominent peak in the area. Also prominent is Monte degli Angeli, 886m high, which stands 5 miles N of the town and has a village, with conspicuous tower, situated near its summit.

Prominent landmarks in the town include a castle, floodlighted at night, standing at the E side; the cupola of the church standing close SW of the castle; and a keep, formed by a truncated cone, situated near the root of the S breakwater of the old

harbor.

An industrial complex extends NE of the town and is brightly lighted at night. It includes large buildings, tanks, and chimneys. A tourist resort complex extends SW of the town.

A light is shown from a white octagonal tower on a two-story building, 18m high, standing near the root of the E breakwater of the old harbor.

Pilotage.—Pilotage is compulsory for vessels over 500 gt and is available 24 hours. Vessels should send a request for a pilot 1 or 2 hours before arrival. The pilot boards in position $41^{\circ}36.0'N$, $15^{\circ}58.0'E$.

Contact Information.—See the table titled **Manfredonia—Contact Information**.

Manfredonia—Contact Information	
Harbormaster	
Call sign	Manfredonia Port Radio
VHF	VHF channel 16
Telephone	39-0884-583-871 39-0884-583-872 39-0884-458-7222
Facsimile	39-0884-587-388
E-mail	manfredonia@guardiacostiera.it cpmanfredonia@mit.gov.it
Web site	http://www.guardiacostiera.gov.it/manfredonia
Port Authority	
Telephone	39-0884-538-547
Facsimile	39-0884-515-635
E-mail	info@autoritaportualemandredonia.com
Tugs	
VHF	VHF channel 12
Telephone	39-0884-541-953
Facsimile	39-0884-581-405
Web site	http://www.federimorchiatori.it
Mooring	
VHF	VHF channels 12, 14, and 16
Telephone	39-0884-533-458
Facsimile	39-0884-533-458
E-mail	ormeggiatorimanfredonia@gmail.com
Pilots	
Call sign	Piloti Manfredonia
VHF	VHF channel 12
Telephone	39-884-584-870 (office hours)
E-mail	pilotimanfredonia@libero.it
Web site	http://www.fedepiloti.it

Figure B.3: Example of a medium table in the document

Imperia—Contact Information	
Web site	http://www.angopi.eu
Pilots	
VHF	VHF channels 12 and 16
Telephone	39-0183-666-345
E-mail	pilotaimperia@alice.it

Anchorage.—Vessels can anchor, in depths of 11 to 13m, sand, good holding ground, in the areas listed below:

1. Anchorage Zone A is a circle, with of radius 500m, centered on position 43°52.7'N, 8°03.0'E. This anchorage is reserved for vessels exceeding 10,000 gt.
2. Anchorage B is a circle of radius 500m, centered 1 mile S of the Porto Mauricio S breakwater head.
3. Anchorage Zone C is a circle, with a radius 500m, centered on position 43°52.1'N, 8°01.6'E.

Restricted areas are established for these anchorages, each with a 500m radius centered on the anchorage center.

The village of Rollo, with a conspicuous belfry, is situated on a hill 1 mile N of Capo Cervo and is prominent from seaward.

Caution.—An anchoring, fishing, and diving prohibited area, the limits of which are shown on the chart, is situated in the vicinity of an historic wreck lying 1.5 miles ESE of Diano Marina.

An outfall extends 0.6 mile SE from Capo Berta.

10.9 Capo Mele (43°57'N., 8°10'E.), located 2.7 miles NE of Capo Cervo, is high and rounded. A light is shown from a prominent tower with a dwelling, 25m high, standing on the cape. A signal station surmounts the summit of the cape.

Currents off this cape are strong and usually set SW. It is reported locally that they attain their greatest velocity between 1 mile and 2 miles offshore. However, at a distance of about 350m from the coast, there may be SW or even a countercurrent at times. The currents are generally not dangerous and tend to set away from the coast.

Caution.—A spoil ground area, with a radius of 0.5 mile, in which anchoring, fishing, diving, and trawling are prohibited,

Figure B.4: Example of a information about a small port

Appendix C

Prompt Template

```
1 DEFAULT_RAG_PROMPT = """\
2 You are an assistant for question-answering tasks about maritime domain.
3     Use the following pieces of retrieved context to answer the question.
4     If you don't know the answer, just say that you don't know. Use
5     three sentences maximum and keep the answer concise.\"
6 """
7
8 QWEN_DEFAULT_RAG_PROMPT = """\
9 You are a knowledgeable assistant specializing in answering questions
10    about ports, sailing, and maritime activities along the Italian
11    peninsula. Use the most relevant pieces of the retrieved context
12    below to support your answer. If you don't know the answer or the
13    context is insufficient, state that you are unsure. Use a maximum of
14    three sentences, and keep your answer concise, accurate, and where
15    applicable, use maritime terminology."""
16
17 phi_rag_prompt_template = f"""<| system|>
18 {DEFAULT_RAG_PROMPT }<| end|>"""+ """
19 <| user|>
20 Question: {input}
21 Context: {context} <| end|>
22 <| assistant|>"""
23
24
25 qwen_rag_prompt_template = f"""<| im_start|>system
26 {QWEN_DEFAULT_RAG_PROMPT }<| im_end|>"""+ """
27 <| im_start|>user
28 Question: {input}
29 Context: {context}
30 <| im_end|>
31 <| im_start|>assistant<| im_end|>
32 """
33
```

```
24 gemma_rag_prompt_template = f"""
25 {DEFAULT_RAG_PROMPT}, """ + """<start_of_turn>user
26 {input}<end_of_turn><start_of_turn>context
27 {context}<end_of_turn>
28 <start_of_turn>model"""
```

Listing C.1: MarineChat Prompt and Response

Appendix D

Evaluation set

User Input	Reference	Response	Retrieved Contexts
What are the typical winds in Genoa?	The typical winds in Genoa are in winter from...	The typical winds in Genoa are predominantly from...	[‘French border) to Genova, then SE and S to Portofino, typical winds are...’]
What is the depth of the sea in the port of Alassio?	The average depth in the port of Alas-sio is 4m	The depth of the sea in the port of Alas-sio ranges...	[“No. 39070), a small port, lies at the SW end...”]
What is the VHF channel of the port of Imperia?	There are two VHF channels in the port of Imperia...	The VHF channels of the port of Imperia are ch...	[‘LAT 0.0m: Predicted heights are in meters...’]
Where is situated the Golfo Marconi?	Marconi Gulf is located between Portofino and...	The Golfo Marconi is situated between Punta di...	[“Golfo Marconi10.20 Golfo Marconi (44°18’N...”]
Can you tell me the email address of the port of Ischia?	Ischia Harbor Email is ischia@guardiacostiera.it uc...	The email address for the port of Ischia	[‘Porto d’Ischia—Contact Information...’]

Table D.1: First five rows of a evaluation set used to evaluate our tool through RAGAs

Bibliography

- [AAA⁺23] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Capelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, et al. The falcon series of open language models. *arXiv preprint arXiv:2311.16867*, 2023.
- [AJA⁺24] Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.
- [AM20] Eleni Adamopoulou and Lefteris Moussiades. Chatbots: History, technology, and applications. *Machine Learning with applications*, 2:100006, 2020.
- [AWW⁺23] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection. *arXiv preprint arXiv:2310.11511*, 2023.
- [BCDP⁺02] Peter Brown, John Cocke, Stephen Della Pietra, Vincent Dellapietra, Fredrick Jelinek, John Lafferty, Robert Mercer, and Paul Roossin. A statistical approach to machine translation. *Computational Linguistics*, 16, 07 2002.
- [Ben75] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975. doi:10.1145/361002.361007.
- [BMP⁺23] Pierpaolo Basile, Elio Musacchio, Marco Polignano, Lucia Siciliani, Giuseppe Fiameni, and Giovanni Semeraro. Llamantino: Llama 2 models for effective text generation in italian language. *arXiv preprint arXiv:2312.09993*, 2023.
- [Bro20] Tom B Brown. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

- [BSY⁺22] Stephen H Bach, Victor Sanh, Zheng-Xin Yong, Albert Webson, Colin Raffel, Nihal V Nayak, Abheesht Sharma, Taewoon Kim, M Saiful Bari, Thibault Fevry, et al. Promptsource: An integrated development environment and repository for natural language prompts. *arXiv preprint arXiv:2202.01279*, 2022.
- [CCB⁺24] Davide Caffagni, Federico Cocchi, Luca Barsellotti, Nicholas Moratelli, Sara Sarto, Lorenzo Baraldi, Marcella Cornia, and Rita Cucchiara. The (r) evolution of multimodal large language models: A survey. *arXiv preprint arXiv:2402.12451*, 2024.
- [CDZ⁺24] Laura Caspari, Kanishka Ghosh Dastidar, Saber Zerhoudi, Jelena Mitrovic, and Michael Granitzer. Beyond benchmarks: Evaluating embedding model similarity for retrieval augmented generation systems. *arXiv preprint arXiv:2407.08275*, 2024.
- [CHL⁺24] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024.
- [CLHS24] Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. Benchmarking large language models in retrieval-augmented generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17754–17762, 2024.
- [CXZ⁺24] Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *arXiv preprint arXiv:2402.03216*, 2024.
- [DCLT19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. URL: <https://arxiv.org/abs/1810.04805>, arXiv:1810.04805.
- [Dev18] Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [DGD⁺24] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvassy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. *arXiv preprint arXiv:2401.08281*, 2024.
- [DHQZ24] Sumit Kumar Dam, Choong Seon Hong, Yu Qiao, and Chaoning Zhang. A complete survey on llm-based ai chatbots. *arXiv preprint arXiv:2406.16937*, 2024.

- [EJEAS23] Shahul Es, Jithin James, Luis Espinosa-Anke, and Steven Schockaert. Ra-gas: Automated evaluation of retrieval augmented generation. *arXiv preprint arXiv:2309.15217*, 2023.
- [FAHA22] Elias Frantar, Saleh Ashkboos, Torsten Hoefer, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.
- [FQY⁺24] Jia Fu, Xiaoting Qin, Fangkai Yang, Lu Wang, Jue Zhang, Qingwei Lin, Yubo Chen, Dongmei Zhang, Saravan Rajmohan, and Qi Zhang. Autorag-hp: Automatic online hyper-parameter tuning for retrieval-augmented generation. *arXiv preprint arXiv:2406.19251*, 2024.
- [FZW⁺24] Hao Fei, Han Zhang, Bin Wang, Lizi Liao, Qian Liu, and Erik Cambria. Empathyear: An open-source avatar multimodal empathetic chatbot. *arXiv preprint arXiv:2406.15177*, 2024.
- [GDW⁺24] Ruihao Gong, Yifu Ding, Zining Wang, Chengtao Lv, Xingyu Zheng, Jinyang Du, Haotong Qin, Jinyang Guo, Michele Magno, and Xianglong Liu. A survey of low-bit large language models: Basics, systems, and algorithms. *arXiv preprint arXiv:2409.16694*, 2024.
- [Gen23] Meta GenAI. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- [GPAM⁺20] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.
- [GZA⁺23] Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, et al. Textbooks are all you need. *arXiv preprint arXiv:2306.11644*, 2023.
- [HBB⁺21] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021. URL: <https://arxiv.org/abs/2009.03300>, arXiv: 2009.03300.
- [HLW23] Yikun Han, Chunjiang Liu, and Pengfei Wang. A comprehensive survey on vector database: Storage and retrieval technique, challenge. *arXiv preprint arXiv:2310.11703*, 2023.

- [HOC⁺24] Musashi Hinck, Matthew L Olson, David Cobbley, Shao-Yen Tseng, and Va-sudev Lal. Llava-gemma: Accelerating multimodal foundation models with a compact language model. *arXiv preprint arXiv:2404.01331*, 2024.
- [HTH⁺24] Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, et al. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024.
- [Hut04] W. John Hutchins. The georgetown-lbm experiment demonstrated in january 1954. In Robert E. Frederking and Kathryn B. Taylor, editors, *Machine Translation: From Real Users to Research*, pages 102–114, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
- [Hut07] John Hutchins. Machine translation: A concise history. *Computer aided translation: Theory and practice*, 13(29-70):11, 2007.
- [HVD15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL: <https://arxiv.org/abs/1503.02531>, arXiv: 1503.02531.
- [HYM⁺23] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *arXiv preprint arXiv:2311.05232*, 2023.
- [JGKS24] Pankaj Joshi, Aditya Gupta, Pankaj Kumar, and Manas Sisodia. Robust multi model rag pipeline for documents containing text, table images. In *2024 3rd International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*, pages 993–999, 2024. doi:10.1109/ICAAIC60222.2024.10574972.
- [JLF⁺23] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Comput. Surv.*, 55(12), March 2023. doi:10.1145/3571730.
- [JZH⁺24] Wenqi Jiang, Shuai Zhang, Boran Han, Jie Wang, Bernie Wang, and Tim Kraska. Piperag: Fast retrieval-augmented generation via algorithm-system co-design. *arXiv preprint arXiv:2403.05676*, 2024.
- [KLS⁺21] Alexander Kozlov, Ivan Lazarevich, Vasily Shamporov, Nikolay Lyalyushkin, and Yury Gorbachev. Neural network compression framework for fast model inference. In *Intelligent Computing: Proceedings of the 2021 Computing Conference, Volume 3*, pages 213–232. Springer, 2021.

- [KMH⁺20] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.
- [KOM⁺20] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*, 2020.
- [Lew19] M Lewis. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.
- [LFL⁺24] Weile Luo, Ruibo Fan, Zeyu Li, Dayou Du, Qiang Wang, and Xiaowen Chu. Benchmarking and dissecting the nvidia hopper gpu architecture. *arXiv preprint arXiv:2402.13499*, 2024.
- [LGF⁺23] Haoran Li, Dadi Guo, Wei Fan, Mingshi Xu, Jie Huang, Fanpu Meng, and Yangqiu Song. Multi-step jailbreaking privacy attacks on chatgpt. *arXiv preprint arXiv:2304.05197*, 2023.
- [LLC⁺24] Zhenyan Lu, Xiang Li, Dongqi Cai, Rongjie Yi, Fangming Liu, Xiwen Zhang, Nicholas D Lane, and Mengwei Xu. Small language models: Survey, measurements, and insights. *arXiv preprint arXiv:2409.15790*, 2024.
- [LLL24] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 26296–26306, 2024.
- [LLY22] Mina Lee, Percy Liang, and Qian Yang. Coauthor: Designing a human-ai collaborative writing dataset for exploring language model capabilities. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI ’22, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3491102.3502030.
- [LPP⁺20] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.
- [LXH⁺24] Jinhao Li, Jiaming Xu, Shan Huang, Yonghua Chen, Wen Li, Jun Liu, Yaoxiu Lian, Jiayi Pan, Li Ding, Hao Zhou, et al. Large language model inference acceleration: A comprehensive hardware perspective. *arXiv preprint arXiv:2410.04466*, 2024.

- [LYF⁺23] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Comput. Surv.*, 55(9), January 2023. doi:10.1145/3560815.
- [LZ01] John Lafferty and Chengxiang Zhai. Document language models, query models, and risk minimization for information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 111–119, 2001.
- [LZI⁺24] Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, et al. Mobilellm: Optimizing sub-billion parameter language models for on-device use cases. *arXiv preprint arXiv:2402.14905*, 2024.
- [MAR24] Seyed Mahed Mousavi, Simone Alghisi, and Giuseppe Riccardi. Is your llm outdated? benchmarking llms & alignment algorithms for time-sensitive knowledge. *arXiv preprint arXiv:2404.08700*, 2024.
- [MdAB⁺13] Maria das Graças Bruno Marietto, Rafael Varago de Aguiar, Gislene de Oliveira Barbosa, Wagner Tanaka Botelho, Edson Pimentel, Robson dos Santos França, and Vera Lúcia da Silva. Artificial intelligence markup language: a brief tutorial. *arXiv preprint arXiv:1307.3091*, 2013.
- [MMN⁺24] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. Large language models: A survey. *arXiv preprint arXiv:2402.06196*, 2024.
- [MTMR22] Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*, 2022. URL: <https://arxiv.org/abs/2210.07316>, doi:10.48550/ARXIV.2210.07316.
- [MY20] Yu A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2020. doi:10.1109/TPAMI.2018.2889473.
- [NC19] Rodrigo Nogueira and Kyunghyun Cho. Passage re-ranking with bert. *arXiv preprint arXiv:1901.04085*, 2019.
- [Omo89] Stephen M Omohundro. Five balltree construction algorithms. 1989.

- [OZHW23] Shuyin Ouyang, Jie M Zhang, Mark Harman, and Meng Wang. Llm is like a box of chocolates: the non-determinism of chatgpt in code generation. *arXiv preprint arXiv:2308.02828*, 2023.
- [PBS24] Marco Polignano, Pierpaolo Basile, and Giovanni Semeraro. Advanced natural-based interaction for the italian language: Llamantino-3-anita. *arXiv preprint arXiv:2405.07101*, 2024.
- [PDC⁺23] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5:606–624, 2023.
- [PLH⁺23] Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277*, 2023.
- [RW97] Stephen E Robertson and Steve Walker. On relevance weights with little relevance information. In *Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 16–24, 1997.
- [RWC⁺19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- [RZ⁺09] Stephen Robertson, Hugo Zaragoza, et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009.
- [Seb23] Glorin Sebastian. Privacy and data protection in chatgpt and other ai chatbots: strategies for securing user information. *International Journal of Security and Privacy in Pervasive Computing (IJSPPC)*, 15(1):1–14, 2023.
- [SEK24] Harald Steck, Chaitanya Ekanadham, and Nathan Kallus. Is cosine-similarity of embeddings really about similarity? In *Companion Proceedings of the ACM on Web Conference 2024*, pages 887–890, 2024.
- [Sen15] Rico Sennrich. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.
- [SFKPZ23] Jon Saad-Falcon, Omar Khattab, Christopher Potts, and Matei Zaharia. Ares: An automated evaluation framework for retrieval-augmented generation systems. *arXiv preprint arXiv:2311.09476*, 2023.

- [SLM⁺24] Sakib Shahriar, Brady D. Lund, Nishith Reddy Mannuru, Muhammad Arbab Arshad, Kadhim Hayawi, Ravi Varma Kumar Bevara, Aashrith Mannuru, and Laiba Batool. Putting gpt-4o to the sword: A comprehensive evaluation of language, vision, speech, and multimodal proficiency. *Applied Sciences*, 14(17), 2024. URL: <https://www.mdpi.com/2076-3417/14/17/7782>, doi: 10.3390/app14177782.
- [SMH⁺24] Bhaskarjit Sarmah, Dhagash Mehta, Benika Hall, Rohan Rao, Sunil Patel, and Stefano Pasquali. Hybriddrag: Integrating knowledge graphs and vector retrieval augmented generation for efficient information extraction. In *Proceedings of the 5th ACM International Conference on AI in Finance*, pages 608–616, 2024.
- [SWLL24] Yifan Song, Guoyin Wang, Sujian Li, and Bill Yuchen Lin. The good, the bad, and the greedy: Evaluation of llms should not ignore non-determinism. *arXiv preprint arXiv:2407.10457*, 2024.
- [Tea24] Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL: <https://qwenlm.github.io/blog/qwen2.5/>.
- [TRP⁺24] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.
- [Tur09] Alan M. Turing. *Computing Machinery and Intelligence*, pages 23–65. Springer Netherlands, Dordrecht, 2009. doi:10.1007/978-1-4020-6710-5_3.
- [TWZ⁺23] Zecheng Tang, Pinzheng Wang, Keyan Zhou, Juntao Li, Ziqiang Cao, and Min Zhang. Can diffusion model achieve better performance in text generation? bridging the gap between training and inference! *arXiv preprint arXiv:2305.04465*, 2023.
- [TZJ⁺24] SM Tonmoy, SM Zaman, Vinija Jain, Anku Rani, Vipula Rawte, Aman Chadha, and Amitava Das. A comprehensive survey of hallucination mitigation techniques in large language models. *arXiv preprint arXiv:2401.01313*, 2024.
- [Vas17] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.
- [VHMN20] Greg Van Houdt, Carlos Mosquera, and Gonzalo Nápoles. A review on the long short-term memory model. *Artificial Intelligence Review*, 53(8):5929–5955, 2020.

- [VNSA⁺24] Chien Van Nguyen, Xuan Shen, Ryan Aponte, Yu Xia, Samyadeep Basu, Zhengmian Hu, Jian Chen, Mihir Parmar, Sasidhar Kunapuli, Joe Barrow, et al. A survey of small language models. *arXiv preprint arXiv:2410.20011*, 2024.
- [Wal03] Richard Wallace. The elements of aiml style. *Alice AI Foundation*, 2003.
- [WCL⁺24] Wenxiao Wang, Wei Chen, Yicong Luo, Yongliu Long, Zhengkai Lin, Liye Zhang, Binbin Lin, Deng Cai, and Xiaofei He. Model compression and efficient inference for large language models: A survey. *arXiv preprint arXiv:2402.09748*, 2024.
- [Wei66] Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.
- [WHW⁺22] Yujing Wang, Yingyan Hou, Haonan Wang, Ziming Miao, Shibin Wu, Qi Chen, Yuqing Xia, Chengmin Chi, Guoshuai Zhao, Zheng Liu, et al. A neural corpus indexer for document retrieval. *Advances in Neural Information Processing Systems*, 35:25600–25614, 2022.
- [WMZ⁺24] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *arXiv preprint arXiv:2406.01574*, 2024.
- [WZS⁺24] Jinheng Wang, Hansong Zhou, Ting Song, Shaoguang Mao, Shuming Ma, Hongyu Wang, Yan Xia, and Furu Wei. 1-bit ai infra: Part 1.1, fast and lossless bitnet b1. 58 inference on cpus. *arXiv preprint arXiv:2410.16144*, 2024.
- [XLM⁺23] Wenhan Xiong, Jingyu Liu, Igor Molybog, Hejia Zhang, Prajjwal Bhargava, Rui Hou, Louis Martin, Rashi Rungta, Karthik Abinav Sankararaman, Barlas Oguz, et al. Effective long-context scaling of foundation models. *arXiv preprint arXiv:2309.16039*, 2023.
- [XLZ⁺23] Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. C-pack: Packaged resources to advance general chinese embedding. *arXiv preprint arXiv:2309.07597*, 2023.
- [YGZ⁺24] Hao Yu, Aoran Gan, Kai Zhang, Shiwei Tong, Qi Liu, and Zhaofeng Liu. Evaluation of retrieval-augmented generation: A survey. *arXiv preprint arXiv:2405.07437*, 2024.

- [YLX⁺24] Biwei Yan, Kun Li, Minghui Xu, Yueyan Dong, Yue Zhang, Zhaochun Ren, and Xiuzhen Cheng. On protecting the data privacy of large language models (llms): A survey. *arXiv preprint arXiv:2403.05156*, 2024.
- [YYH⁺21] Xi Ye, Semih Yavuz, Kazuma Hashimoto, Yingbo Zhou, and Caiming Xiong. Rng-kbqa: Generation augmented iterative ranking for knowledge base question answering. *arXiv preprint arXiv:2109.08678*, 2021.
- [YYH⁺24] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.
- [YYZ⁺24] Yuan Yao, Tianyu Yu, Ao Zhang, Chongyi Wang, Junbo Cui, Hongji Zhu, Tianchi Cai, Haoyu Li, Weilin Zhao, Zhihui He, et al. Minicpm-v: A gpt-4v level mllm on your phone. *arXiv preprint arXiv:2408.01800*, 2024.
- [YZS⁺23] Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. Diffusion models: A comprehensive survey of methods and applications. *ACM Computing Surveys*, 56(4):1–39, 2023.
- [ZDL⁺23] Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Fei Wu, et al. Instruction tuning for large language models: A survey. *arXiv preprint arXiv:2308.10792*, 2023.
- [ZLC⁺23] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. Siren’s song in the ai ocean: a survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219*, 2023.
- [ZLCL23] Zhengping Zhou, Lezhi Li, Xinxi Chen, and Andy Li. Mini-giants:” small” language models and open source win-win. *arXiv preprint arXiv:2307.08189*, 2023.
- [ZWC⁺24] Hailin Zhang, Yujing Wang, Qi Chen, Ruiheng Chang, Ting Zhang, Ziming Miao, Yingyan Hou, Yang Ding, Xupeng Miao, Haonan Wang, et al. Model-enhanced vector index. *Advances in Neural Information Processing Systems*, 36, 2024.
- [ZZY⁺24] Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, and Bin Cui. Retrieval-augmented generation for ai-generated content: A survey. *arXiv preprint arXiv:2402.19473*, 2024.