POLITECNICO DI TORINO

Master's degree in ICT FOR SMART SOCIETIES

in collaboration with ARGOTEC

Optimizing Retrieval-Augmented Generation for Space Mission Design via Multi-Task Learning

Supervisors
Alessandro BALOSSINO
Silvia BUCCI
Luca CAGLIERO
Edoardo FADDA

Candidate
Federico VOLPONI
309709

October 2024

Abstract

The extensive documentation required during the early phases of space mission design, which must be reviewed and analyzed by systems engineers, often slows down the design process. Given the time-intensive nature of this stage, integrating a Virtual Assistant (VA) can significantly enhance the efficiency of information retrieval. Recent advancements in Natural Language Processing (NLP), particularly the development of Large Language Models (LLMs), have created opportunities for non-invasive question-answering VAs. However, the limited technical knowledge of LLMs in the space domain and their lack of up-to-date information pose significant challenges for real-world applications. Retrieval-Augmented Generation (RAG) offers a solution by incorporating an external knowledge source, from which a retriever selects the most relevant information based on the user's query. In this thesis, we enhance the retriever component by fine-tuning it on space-domain data using supervised and Multi-Task Learning (MTL) approaches, demonstrating how improvements in the retriever impact the performance of the generator. In the MTL setting, we combined supervised with self-supervised tasks showing how the network benefits from the addition of complementary tasks. Finally, we tested the enhanced RAG pipeline during a Concurrent Engineering (CE) session at the Argotec Advanced Concept Laboratory (ACLab), yielding promising results in a real-world scenario.

Keywords: Large Language Models, Multi-Task Learning, Information Retrieval, Concurrent Engineering

Contents

Acronyms

ACLab Advanced Concepts Laboratory.

AI Artificial Intelligence.

ANN Approximate Nearest Neighbor.

BGE BAAI General Embedding.

CA Cross-attention.

CE Concurrent Engineering.

CT Contrastive Tension.

HNSW Hierarchical Navigable Small World.

IR Information Retrieval.

KB Knowledge Base.

LLM Large Language Model.

MHA Multi-head Attention.

MLM Masked Language Modeling.

MNRL Multiple Negative Ranking Loss.

MTAL Multi-Task Auxiliary Learning.

MTL Multi-Task Learning.

NLP Natural Language Processing.

NSP Next Sentence Prediction.

OCR Optical Character Recognition.

PLM Pre-trained Language Model.

RAG	Retrieval-augmented Generation.
SA	Self-attention.
SWA	Sliding Window Attention.
VA	Virtual Assistant.

List of Figures

List of Tables

Chapter 1

Introduction

Space mission design is a challenging problem for several reasons; among those, there is a substantial volume of documentation involved in the design process that has to be analyzed and studied by systems engineers. This task is strongly time-consuming but essential during trade-space exploration and early phase design. To optimize this process, different studies [13, 18, 35] have proposed virtual assistants (VAs), which are helpful in knowledge management of the aerospace sector, leveraging Machine Learning and Natural Language Processing (NLP) methods.

The NLP field has garnered significant attention in recent years, primarily due to the rise of Large Language Models (LLMs). These advanced models demonstrate exceptional capabilities in understanding and generating human-like text, and their versatility extends to solving a wide range of tasks. Thus LLMs are becoming the first building block for creating an artificial general intelligence that can take to the next step the development of new VAs. These VAs can provide context-aware, precise, and efficient support, thereby significantly enhancing productivity and decision-making processes.

One sector where LLM-based virtual assistants could make a substantial impact is within the framework of concurrent engineering (CE), a methodology that is becoming increasingly prevalent in space mission design. CE facilitates efficient and effective mission design through real-time collaborative design sessions involving multi-disciplinary teams. This stands in contrast to the traditional sequential approach, which often leads to delays, and inconsistencies: indeed, in the conventional method, each specialist develops a subsystem design independently, which restricts the potential for interdisciplinary solutions and significantly increases the time required to conduct comprehensive studies.

The significance of a robust software infrastructure in the concurrent engineering approach cannot be neglected. According to [2], having a well-integrated software system is a critical component of CE. Within this context, the integration of a virtual assistant can add substantial value to the process, particularly as a documentation support system. Given the vast amount of internal and external documentation involved in space mission design, investing in an efficient Information Retrieval (IR) system becomes crucial. Such a system would enable quick and precise searches for specific information, thereby streamlining the design process and enhancing the overall efficiency of concurrent engineering efforts.

Argotec, an Italian aerospace engineering company, is currently implementing a CE framework, called the Advanced Concepts Laboratory (ACLab) to improve the efficiency and effectiveness of its processes for mission formulation and design. Along with building the ACLab facility, Argotec is investing in novel approaches to further improve the efficiency of the design process; among those, there is the development of a Retrieval-Augmented Generation (RAG) [26] pipeline to support IR during the trade-space exploration and early phase design and the design of a task prioritization approach to estimate the time required for various tasks and plan the CE sessions.

RAG addresses several key issues of standalone LLMs, including the problem of hallucination, the need for domain-specific accuracy, and the requirement for up-to-date information. It works by combining a retriever model, which searches a Knowledge Base (KB) to find relevant documents, with a generator model that uses this retrieved information to produce more accurate and contextually appropriate responses. This approach ensures that the generated text is grounded in real data, enhancing reliability and relevance, especially in specialized fields like aerospace.

Figure 1.1 shows the high-level workflow of the ACLab methodology, together with the task prioritization and RAG integrations. Before starting the study, the ACLab workflow requires iterations with the client to consolidate or define the study's objectives and constraints. This step is crucial for a better understanding of the client's needs and to avoid studies with unclear objectives. Once the study is approved, a study lead is assigned. The study lead is responsible for understanding the client's requests, preparing the study, and guiding discussions during the sessions. During this preparatory phase, the study lead identifies the required areas of expertise, assembles the team, and plans the sessions. A task prioritization approach is used to help the study lead plan the sessions and allocate resources effectively. After the study preparation, the design sessions can begin. The preparatory phase is concluded with the mission objectives flow-down, while the first in-session stage corresponds to trade space exploration, in particular concepts generation and selection. During this phase, engineers can leverage the RAG knowledge base to quickly find information useful for the design. Finally, the point design phase consists of the actual design of the previously explored concepts. It is relevant to note that the study report produced at the end of the CE sessions is given in input to the KB, keeping the architecture knowledge constantly up-to-date.

The aim of this thesis is the development of a RAG pipeline tailored for space-related data, with particular emphasis on improving the retriever

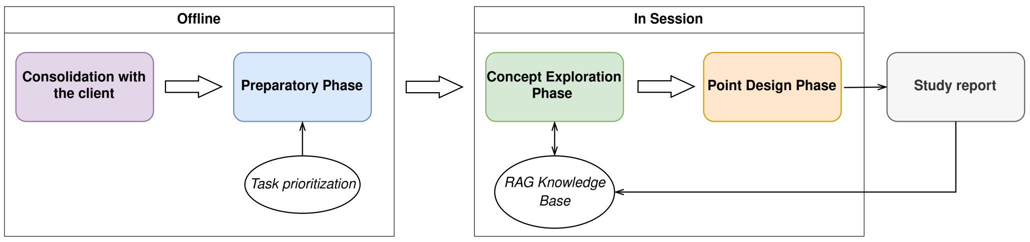


Figure 1.1: ACLab high-level methodology together with the task prioritization and RAG integrations.

component by fine-tuning the embedding model for domain adaptation in the aerospace sector. Besides the traditional supervised and self-supervised methods, we have also explored multi-task learning approaches. The work is organized as follows: in chapter 2 related works on the NLP domain and on the development of previous VAs in the space domain are discussed; chapter 3 describes in detail how the RAG pipeline works, starting from an overview of the working principle of transformer-based models, and then focusing on the retrieval and generation stages; chapter 4 is dedicated to fine-tuning of the embedding model; chapter 5 shows the experimental setup, the experiments and the obtained results; finally, in the last chapter the conclusions are drawn.

Chapter 2

Related works

This chapter focuses on a description of related works; starting with the analysis of previous VAs in the space domain, we move to an overview of the recent advancements in the NLP fields and then we focus on fine-tuning methods of embedding models.

2.1 AI for Space Mission Design

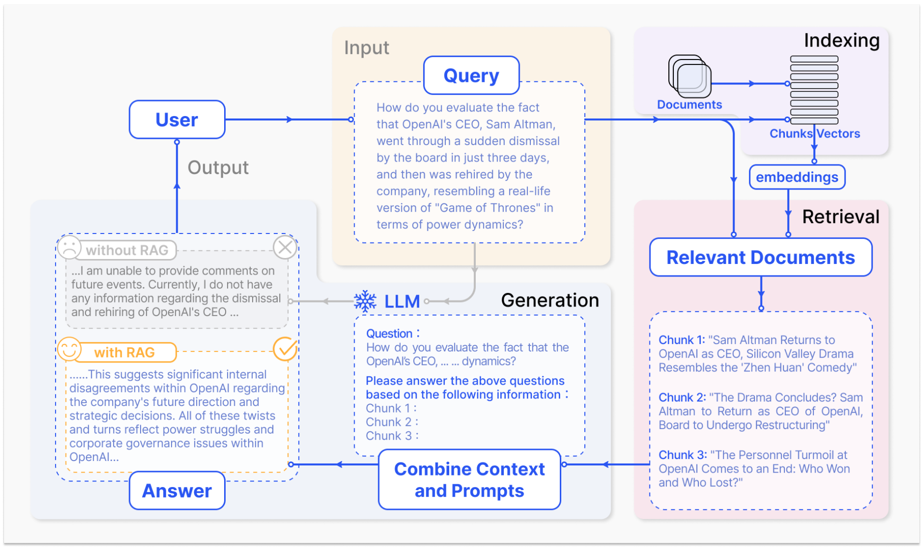
The integration of Artificial Intelligence (AI) in the design and analysis of space missions is a growing area of research, particularly focusing on improving the efficiency and capabilities of CE. Several projects have explored the use of AI-powered virtual assistants to aid engineers during spacecraft mission design. Among those, the most recent and promising are Daphne [18], the Design Engineering Assistant (DEA) [13] and SpaceQA [35]. Daphne is a virtual assistant for designing Earth observation distributed spacecraft missions. Its comprehensive question-answering system and cognitive assistance features were assessed through a study at NASA's JPL involving nine people. The findings suggest that Daphne can improve performance during system design tasks compared to traditional tools. The system follows a microservices approach and it consists of three main parts: a web front-end, a front-end server (called the Daphne Brain) that directs all user requests to the third part, the roles, which are software snippets that interface with different backends and data sources. In [13] is shown how the field of knowledge representation and reasoning can be successfully employed at the start of the space mission life cycle via an expert system used as a DEA, especially during CE sessions. It emphasizes the utility of converting unstructured data into structured data to be stored into a knowledge graph; the KG can be used by an inference engine to support the generation of initial design inputs and provide easy and quick access to previous designs. This study makes use of ontology learning methods to automate the knowledge base generation and population. SpaceQA is an open-domain question-answering system specifically designed for space mission design. Developed under ESA initiative, SpaceQA utilizes an architecture combining a dense retriever and a neural reader, following the same workflow of a RAG pipeline. Due to the scarcity of domain-specific annotated data, the work relies on transfer learning rather than fine-tuning. Preliminary evaluations indicate the effectiveness of this generative IR system in facilitating access to information about space mission design.

Collectively, these works underscore the transformative potential of AI-driven tools in streamlining and augmenting the complex processes involved in space mission design, paving the way for more efficient and informed decision-making within concurrent design facilities.

2.2 Transformer-based Models and Augmentation

The invention of the transformer architecture [10] marked an important milestone in the NLP field. Indeed, with respect to previous neural language models [6], [8], [7], allows more parallelization making it possible to pre-train very big language models on a large amount of data on GPUs. Transformer relies entirely on self-attention enabling the capture of long-range dependencies between tokens.

The Transformer-based Pre-trained Language Models (PLMs) can be grouped into three categories based on the arrangement of the encoder and decoder components. The *encoder-decoder*, or *sequence-to-sequence* category, presented in the original paper to tackle machine translation tasks, takes the input sequence from the encoder and produces a fixed-size vector representation of it, which is fed into the decoder to generate the output sequence. Additionally to self-attention, the decoder uses cross-attention to apply the attention mechanism to the output of the decoder and the input of the decoder. Popular PLMs in this category are T5 [27] and BART [16]. *Encoder-only*, or *autoencoding* models, are primarily used for language understanding tasks, such as text-classification and sentence-pair regression tasks. The most popular transformer model of this category is BERT [11], which is the starting point of other variants such as RoBERTa [17], ALBERT [15], and DeBERTa [25]. Unlike the encoder-decoder architecture, the transformer encoder is only concerned with the input sequence and does not generate any output sequence. It applies self-attention mechanism to the input tokens, allowing it to focus on the most relevant parts of the input for the given task. Finally, *decoder-only*, or *autoregressive* models, are suitable for text generation tasks since they leverage on prior tokens to predict the next token iteratively. The most widely used models of this category are the ones of the GPT family ([14], [19], [23], [40]). Among the autoregressive models, are worth mentioning the open-source LLaMA and Mistral families which achieve comparable results with the proprietary models of OpenAI, allowing domain-specific research and usage from companies that are constrained by privacy requirements.



How do you evaluate the fact that OpenAI's CEO, Sam Altman, went through a sudden dismissal by the board in just three days, and then was rehired by the company, resembling a real-life version of "Game of Thrones" in terms of power dynamics?

...I am unable to provide comments on future events. Currently, I do not have any information regarding the dismissal and rehiring of OpenAI's CEO ...

......This suggests significant internal disagreements within OpenAI regarding the company's future direction and strategic decisions. All of these twists and turns reflect power struggles and corporate governance issues within OpenAI...

Question :
How do you evaluate the fact that the OpenAI's CEO, ... dynamics?
Please answer the above questions based on the following information :
Chunk 1 :
Chunk 2 :
Chunk 3 :

Chunk 1: "Sam Altman Returns to OpenAI as CEO, Silicon Valley Drama Resembles the 'Zhen Huan' Comedy"

Chunk 2: "The Drama Concludes? Sam Altman to Return as CEO of OpenAI, Board to Undergo Restructuring"

Chunk 3: "The Personnel Turmoil at OpenAI Comes to an End: Who Won and Who Lost?"

Figure 2.1: A high-level overview of the RAG process applied to question answering [42].

Even if the most recent LLMs proved high capabilities in terms of world knowledge they are still affected by limitations, that depending on the use case, require augmentation with some external means. Indeed, LLMs have stale information meaning that their knowledge is limited to the data present in the training set. As a result, their knowledge is limited to the time when they were trained. Moreover, LLMs are trained to predict the next token from a probability distribution without sensing the truthfulness of the generated output, resulting in plausible but misleading answers: this phenomenon is called "hallucination". This problem has been deeply analyzed in [43].

To address the limitation of the lack of up-to-date knowledge or access to private or specific information, particularly useful for enterprise usage of LLMs, RAG comes into play [26]. An overview of RAG is shown in Figure 2.1.

The idea behind RAG is to combine the parametric memory of a PLM with an external source of non-parametric memory. The process involves extracting a query from the input prompt and using that query to retrieve relevant information from an external knowledge source. The relevant information is extracted based on the semantic similarity between the sentence embeddings, created by a sentence transformer model [20], of the query and the documents in the knowledge base. The retrieved relevant information is then added to the original prompt and fed to the generator in order to generate the final response. This approach is the most simplistic one and is defined Naive RAG [42].

From the naive implementation [59] reports several enhancements that can be applied to the different components of the pipeline: input, retriever, generator, and the entire pipeline. When RAG is used on domain-specific and technical data (e.g. medical, space, biology, etc) it is worth performing a fine-tuning of the architecture. In [26] the pipeline is trained end-to-end, specifically fine-tuning the query encoder and the generator. In [47] is explored a joint training of the entire retriever (query encoder and passage encoder) showing promising results but with a high computational cost. However, they achieved improvements by fine-tuning the retriever alone on domain-specific data. Given the high performances and the big size of the latest LLMs, training the entire pipeline is computationally demanding; for this reason, focusing on the fine-tuning of the embedding model used in the retrieval stage on domain-specific data is a reasonable trade-off between computational cost and improvement of the pipeline.

2.3 Fine-tuning of embedding models

The most used approaches for fine-tuning embedding models heavily rely on labeled data. In [20] propose three different objectives to update the model weights such that the produced sentence embeddings are semantically meaningful. The classification objective function concatenates two sentence embeddings with their element-wise difference multiplied by the trainable weight and finally optimizes the cross-entropy loss; the regression objective function computes the cosine-similarity between two sentence embeddings and uses the mean-squared error loss as the objective function to be minimized; finally, the triplet objective function is based on an anchor, a positive and a negative sentence and minimizes the distance between the anchor and the positive such that their distance is smaller than the distance between the anchor and the negative sentence. In [9] present an effective loss function, called Multiple Negative Ranking Loss (MNRL), when working only on positive pairs; if training an embedding model for retrieval the positive pairs can be the query and the relevant passage.

Although the supervised learning approach is the most effective, the scarcity of domain-specific labeled data makes it necessary to leverage self-supervised training methods. One of the most popular is Masked Language Modeling (MLM), the same task used to pre-train BERT, which consists of minimizing the cross-entropy loss on predicting randomly masked tokens. In [33] is shown the effectiveness of MLM for domain adaptation and presents TSDAE that works as a reconstruction task: a noise is added to the input sequence, like the deletion of random words, then the damaged sentences are encoded, and then reconstructed into the original form. SimCSE [31] explores a contrastive learning approach that takes an input sequence and predicts itself in a contrastive objective, with only standard dropout used as noise.

Finally, [29] introduces a method called Contrastive Tension (CT). CT builds two independent encoders initialized with the same weights and uses them to encode sentences. The method is based on the finding that if the two models encode the same sentence then the dot product should be large, and small otherwise.

An alternative approach that is deeply analyzed in [30] is Multi-Task Learning (MTL); of particular interest for the purpose of this thesis is auxiliary MTL where auxiliary tasks are introduced to improve the performance of a primary task. Moreover, the survey shows different methods for the loss construction for MTL; for example, proposes to combine losses by assigning different weights to each of them, making some tasks prevalent compared to others.

Another way to address the data scarcity problem in domain-specific scenarios is proposed by [49]; they show how high-quality text embeddings can be obtained by training a model using only synthetic data. This approach has been widely adopted in recent years thanks to the capabilities of the latest LLMs, in particular, proprietary ones, that can generate high-quality synthetic data.

Chapter 3

Retrieval-Augmented Generation

In this chapter, we analyze in depth the working principles of the RAG pipeline, dividing the workflow into the retrieval and generation phases, and we review the transformer architecture being the core component of the Language Models used in this thesis.

3.1 Overview of the RAG pipeline

Figure 3.1 depicts a high-level overview of the RAG workflow, for both the offline and online stages.

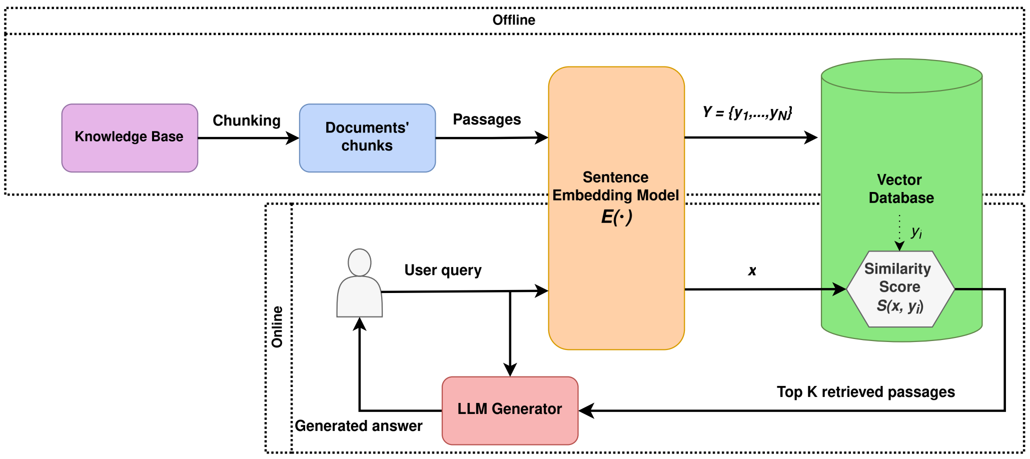


Figure 3.1: The workflow of the RAG process, highlighting the offline and online stages.

The offline stage starts by selecting the set of documents that constitute the Knowledge Base (KB); those can have different formats (e.g. PDF, HTML, PowerPoint, Markdown, etc.) which are then converted into a uniform plain text format. After, each document is segmented into smaller chunks in order to respect the context limitation of the used language models, but also to reduce the impact of the inference speed due to a large input and to make more accessible to the user the context that the generator uses for generating the response. All the N chunks, or passages, are then input into an embedding model $E(\cdot)$ which creates semantically meaningful vector representations $Y = \{y_1, y_2, ..., y_N\}$ ready to be stored in a vector database.

The online stage begins with the user query being encoded with the same embedding model $E(\cdot)$. The resulting query embedding x and the encoded passages y_i are then used to compute the similarity score $S(x, y_i)$, $\forall i \in [1, N]$ (the most used similarity scores are the cosine similarity and the dot product). Finally, the top k passages for similarity score are retrieved and, together with the original user query, given in input to an LLM, which has the role of the generator. The generator is prompted to answer the query considering only the retrieved passages.

3.2 Transformer-based models

All the models that build the RAG method are based on the transformer architecture proposed by [10], or of one of its variations. The original transformer architecture was introduced for sequence-to-sequence tasks, e.g. machine translation; for this reason, it is formed by two components, the encoder and the decoder. Considering that the decoder adds only some variations to the encoder, we start by describing the encoder's functionality and then point out the differences between the two architectures.

3.2.1 The Attention Mechanism

Before diving into the structure of the encoder it is necessary to introduce the concept of attention, being the most important building block of transformer models. We start by denoting as $\boldsymbol{X} \in \mathbb{R}^{n \times e}$ a sentence composed of n tokens embedded in e-dimensional vectors and define as $\boldsymbol{x}_i$ the token at position i. Then we define three matrices, composed by trainable weights, $\boldsymbol{W}_k \in \mathbb{R}^{k \times e}$, $\boldsymbol{W}_v \in \mathbb{R}^{v \times e}$, and $\boldsymbol{W}_q \in \mathbb{R}^{k \times e}$, where k and v are chosen hyperparameters. By using these matrices we can define the key, value, and query tokens:

$$\boldsymbol{k}_i = \boldsymbol{W}_k \boldsymbol{x}_i \tag{3.1}$$

$$\boldsymbol{v}_i = \boldsymbol{W}_v \boldsymbol{x}_i \tag{3.2}$$

$$\boldsymbol{q}_i = \boldsymbol{W}_q \boldsymbol{x}_i \tag{3.3}$$

Then, the attention scores $\boldsymbol{h}_i$ for token $\boldsymbol{x}_i$ are defined as:

$$\boldsymbol{h}_i = \sum_{j=1}^{n} \text{softmax}_j(g(\boldsymbol{q}_i, \boldsymbol{k}_j))\boldsymbol{v_j} \tag{3.4}$$

where $g(\cdot)$ is referred to as the attention scoring function that defines a measure of similarity between tokens and softmax is defined as:

$$[\text{softmax}(\boldsymbol{z})]_i = \frac{\exp(z_i)}{\sum_{j=1}^{N} \exp(z_j)}. \tag{3.5}$$

The attention scoring function is usually expressed as the dot product:

$$g(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^\intercal \boldsymbol{x}_j. \tag{3.6}$$

The reason for using the softmax is to have values between 0 and 1 and to ensure nonnegative values. For completeness the softmax definition for a generic vector $\boldsymbol{z} \in \mathbb{R}^N$ is defined at 3.5.

If we use the dot-product as the attention scoring function we can rewrite the key, value, and query in matrix form for all tokens:

$$\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_k \tag{3.7}$$

$$\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_v \tag{3.8}$$

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_q \tag{3.9}$$

and define the Self-Attention layer (SA) for $\boldsymbol{X}$ as:

$$\text{SA}(\boldsymbol{X}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\intercal}{\sqrt{k}}\right)\boldsymbol{V} \tag{3.10}$$

where we assume the softmax is applied row-wise. In [10] Equation 3.10 is defined as scaled dot-product attention. The scaling factor $\sqrt{k}$ is inserted to ensure that the variance of the dot-product remains at 1 regardless the size of k.

Having just one SA layer allows modeling one single set of dependencies across tokens. However, if we think about a complex sentence in natural language it is easy to understand that one word could have different relationships with other words; hence, we want to have different SA layers that work in parallel, forming a Multi-Head Attention layer (MHA). The number of heads (i.e., the SA layers), is controlled by the hyper-parameter h. The MHA is simply obtained by concatenating the h heads of SA. When working with MHA we need to define an additional matrix of trainable weights, $\boldsymbol{W}_o \in \mathbb{R}^{e \times hv}$, which is used for the concatenation of the h heads. At the end, the MHA layer for input $\boldsymbol{X}$ is defined as:

$$\text{MHA}(\boldsymbol{X}) = [\,\text{SA}_1(\boldsymbol{X}) \,||\, \text{SA}_2(\boldsymbol{X}) \,||\, \dots \,||\, \text{SA}_h(\boldsymbol{X})\,]\,\boldsymbol{W}_o \tag{3.11}$$

where each key, query, and value matrix of each SA layer has its own matrix of trainable weights.

3.2.2 Positional Embeddings

Before describing the entire encoder we need to define an additional component: positional embeddings. Indeed, we need a way to take into consideration the order of the sequence, by inserting some information about each token absolute position or relative position of tokens. In [10] the author use deterministic absolute position encodings of dimensionality e, such that can be summed with the embeddings. So, the operation that is performed before entering the MHA layer is:

$$\boldsymbol{X}' = \boldsymbol{X} + \boldsymbol{P} \tag{3.12}$$

where $\boldsymbol{P} \in \mathbb{R}^{n \times e}$ are the positional embeddings. In the original transformer paper are introduced sinusoidal position embeddings, which for token i are defined as:

$$\boldsymbol{S}_i = \left[\sin(\omega_1 i), \cos(\omega_1 i), \sin(\omega_2 i), \cos(\omega_2 i), \ldots, \sin(\omega_{e/2} i), \cos(\omega_{e/2} i)\right] \tag{3.13}$$

where $\omega_i = \frac{1}{10000^{i/e}}$.

3.2.3 The Transformer

Finally, The encoder schema is shown in Figure 3.2. The Feed-Forward

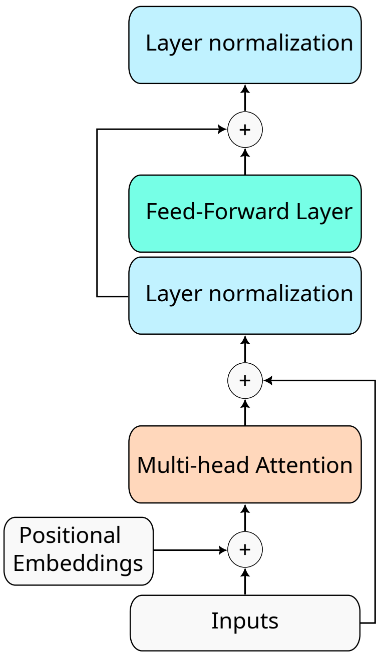


Figure 3.2: The encoder block architecture.

Layer is defined as:

$$\text{FFN}(\boldsymbol{x}) = \text{ReLU}(\boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2, \quad \text{ReLU}(x) = \max(0, x) \tag{3.14}$$

where $\boldsymbol{x} \in \mathbb{R}^e$ is a token embedding, $\boldsymbol{W}_{1,2} \in \mathbb{R}^{p\times e}$, and $\boldsymbol{b}_{1,2} \in \mathbb{R}^e$, are trainable parameters. While layer normalization for a generic vector $\boldsymbol{a} \in \mathbb{R}^n$ is defined as:

$$\text{LayerNorm}(\boldsymbol{a}) = \gamma\frac{\boldsymbol{a} - \mu}{\sigma} + \beta \tag{3.15}$$

where μ and σ are the mean and standard deviation of $\boldsymbol{a}$ and γ and \$*beta* are learnable parameters. In the end, what happens inside the encode block is the following:

1. Define an input sequence of n tokens as $\boldsymbol{X} \in \mathbb{R}^{n\times e}$.
2. Add the positional encodings to the input: $\boldsymbol{X}' = \boldsymbol{X} + \boldsymbol{P}$.
3. Compute the MHA on $\boldsymbol{X}'$ with formula 3.11
4. Add a residual connection to the MHA output and pass it to the normalization layer.
5. Move the output to the FFN layer as in 3.14.
6. Finally, add the second residual connection and perform normalization again.

Note that at the end of the encoder block is usually added an additional head depending on the task, eg. classification.

Note that even if the original transformer architecture is an encoder-decoder model, it is possible to have encoder-only models that follow the same working principle described above.

Now we can proceed by describing the changes needed to pass from the encoder to the decoder architecture. Indeed, the decoder must have a way to condition is output based on the encoder's output and must be causal, i.e. at each step its output depends only on previous elements, in order to perform autoregression. The first addition is addressed by introducing cross-attention (CA). Let's define $\boldsymbol{H} \in \mathbb{R}^{n\times e}$ the output of the encoder's block and $\boldsymbol{X} \in \mathbb{R}^{n\times e}$ the input of the decoder; then the CA between $\boldsymbol{X}$ and $\boldsymbol{H}$ is:

$$\text{CA}(\boldsymbol{X}, \boldsymbol{H}) = \text{softmax}\left(\frac{\boldsymbol{X}\boldsymbol{W}_q\boldsymbol{W}_k^\intercal\boldsymbol{H}^\intercal}{\sqrt{k}}\right)\boldsymbol{H}\boldsymbol{W}_v = \text{softmax}\left(\frac{\boldsymbol{Q}_X\boldsymbol{K}_H^\intercal}{\sqrt{k}}\right)\boldsymbol{V}_H \tag{3.16}$$

The interpretation is that the embeddings $\boldsymbol{X}$ are updated based on their similarity with keys and values provided by $\boldsymbol{H}$. Note that if we want to build a decoder-only model we do not need a CA layer. The second problem of making the decoder block causal is solved by properly masking the MHA, so that the prediction of token i depends only on tokens in positions less

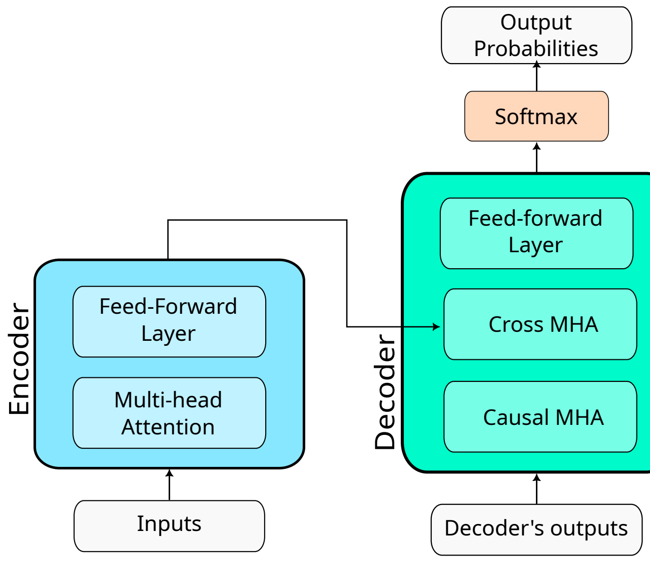


Figure 3.3: Simplified version of the transformer architecture.

than i, i.e. we do not consider "future" tokens. We can accomplish this by defining a mask matrix $\boldsymbol{M} \in \mathbb{R}^{n \times n}$, where:

$$M_{i,j} = \begin{cases} -\infty & \text{if } i > j \\ 0 & \text{otherwise} \end{cases} \tag{3.17}$$

, and then multiplying the mask matrix element-wise with softmax's argument numerator in SA:

$$\text{Masked-SA}(\boldsymbol{X}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^{\intercal} \odot \boldsymbol{M}}{\sqrt{k}}\right)\boldsymbol{V} \tag{3.18}$$

Once, the two components to make the decoder causal and dependant from the encoder's output, we can put together what we have seen so far and build the entire transformer architecture, as depicted in Figure 3.3

3.3 Retrieval phase

The retrieval phase of the RAG pipeline is responsible for fetching relevant information from a large corpus of data using as the only source the user's prompt. In order to compute the relevancy between the user's query and the documents chunks, or passages, we need a function $f(\cdot)$ that given a query q and the passages p_i , $\forall i \in N$, where N is the total number of passages, computes a meaningful vector representation of them, such that they can be compared. Based on the type of function $f(\cdot)$ that is used we can categorize dense and sparse retrievers. Dense retriever methods represent queries and passages using dense embedding vectors, i.e. mainly composed of non-zero values, and are typically computed using transformer-based PLM, like BERT; on the other side, sparse retriever methods leverage word statistics and lexical matching to project a text to a sparse vector, i.e. most of the elements are zero: among those, there are methods like TF-IDF [1] and BM25 [5]. Since the dense retriever models used in this work are based on BERT, an overview of the model is proposed in the following.

3.3.1 BERT Architecture

BERT, which stands for Bidirectional Encoder Representations from Transformers, is an encoder-only transformer based on the implementation of the original transformer model that has been discussed in section 3.2. The bidirectionality of BERT comes from the usage of the encoder only which leverages the entire self-attention to capture the relationship between all the tokens in the context. One of the novelties introduced by BERT is about the input and output representations. Indeed, *Devlin et al.* introduce a special classification token `[CLS]` at the beginning of every sentence that, in its final hidden state, is used to aggregate sequence representation for classification tasks. Moreover, the special token `[SEP]` is used to separate two sentences that are packed together into a single sequence. Since the model can process multiple sentences divided by the `[SEP]` token, they introduced a new type of embeddings, segment embeddings $\boldsymbol{S} \in \mathbb{R}^{\boldsymbol{n} \times \boldsymbol{e}}$, that indicates to which sentence a token belongs to. Segment embeddings add up with the token and position embeddings; then, Equation 3.12 can be written as:

$$\boldsymbol{X}' = \boldsymbol{X} + \boldsymbol{P} + \boldsymbol{S} \tag{3.19}$$

To pre-train the bidirectional model they leverage the Masked Language Modeling (MLM) procedure combined with Next Sentence Prediction (NSP). MLM is needed to train a bidirectional model because the task of predicting the next word is not feasible since the model's bidirectionality allows each token to know which the next word is, making the prediction of the target trivial. Instead, in MLM a percentage of the input sentence tokens is replaced with a `[MASK]` token and then the final hidden state is used to predict the original token with cross-entropy loss. However, since the

`[MASK]` token does not appear during fine-tuning, to mitigate the mismatch between pre-training and fine-tuning, only 80% of the token chosen for prediction is replaced with the special token, while a 10% is replaced with a random token, and the remaining ones are left unchanged. The goal of NSP is to make the model understand sentence relationships. Specifically, the task consists of labeling the sentence after the `[SEP]` token as the next or not the next sentence and using the last hidden state of the `[CLS]` token for prediction.

In addition to pre-training the model can be fine-tuned on specific downstream tasks, eg. question answering, named entity recognition, starting from the pre-trained model parameters. Figure 3.4 shows a high-level overview of the pre-training and fine-tuning tasks.

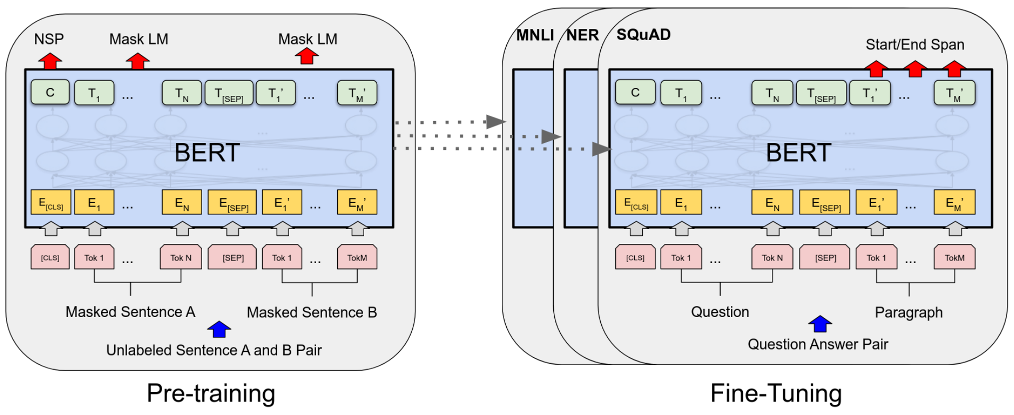
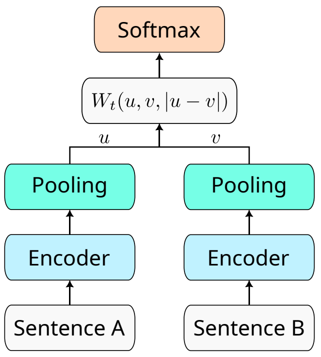


Figure 3.4: Pre-training and fine-tuning procedures for BERT [11]
.

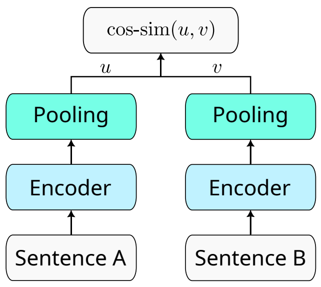
3.3.2 Sentence Transformers

For the retriever purpose, it is needed an embedding that contains the semantic meaning of an entire sentence; from the BERT model, it is trivially achieved by either averaging the output word embeddings or by using the `[CLS]` token as the vector representation of the sentence: this operation is called pooling. However, this approach alone yields rather bad results; for this reason, [20] presented Sentence-BERT, a bi-encoder architecture of the BERT network using siamese and triplet networks, in order to fine-tune the base model, such that the produced sentence embeddings are semantically meaningful and can be compared with a similarity function (eg. cosine-similarity). Additionally, to the previously described pooling methods, they tried also to compute the max-over-time of the output vectors. However, from their experiments, they found that computing the average of the word embeddings yields the best results overall.

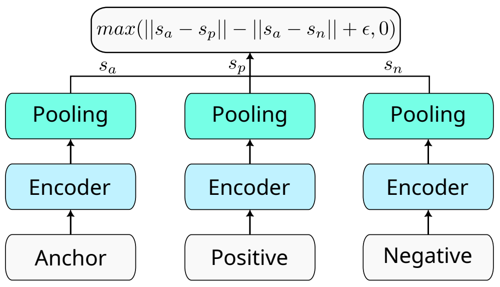
The proposed structures and objective functions are shown in Figure 3.5. In the Classification Objective function, the two sentence embeddings $\boldsymbol{u}$ and



(a) Classification Objective Function



(b) Regression Objective Function



(c) Triplet Objective Function

Figure 3.5: Training objective functions for Sentence-BERT.

$\boldsymbol{v}$ are concatenated with the element-wise difference $|u - v|$ and multiplied by a trainable matrix $\boldsymbol{W}_t \in \mathbb{R}^{3n \times k}$, where n is the dimension of the sentence embedding and k is the number of labels. Then the output is given in input to the softmax:

$$o = \text{softmax}(\boldsymbol{W}_t(\boldsymbol{u}, \boldsymbol{v}, |\boldsymbol{u} - \boldsymbol{v}|)) \tag{3.20}$$

and optimized using the cross-entropy loss. The Regression Objective Function simply computes the cosine-similarity of the sentence embeddings and minimizes the mean-squared error loss. Finally, the Triplet Objective Function leverages three sentences: an anchor sentence a, a positive sentence p, and a negative sentence n. The goal is to tune the network by reducing the distance between a and p such that it becomes smaller than the distance between a and n. Given the sentence embeddings $\boldsymbol{s}_{a|p|n}$ the objective is to minimize the following:

$$max(||\boldsymbol{s}_a - \boldsymbol{s}_p|| - ||\boldsymbol{s}_a - \boldsymbol{s}_n|| + \epsilon, 0) \tag{3.21}$$

where ϵ is a margin which ensures that $\boldsymbol{s}_p$ is at least ϵ closer to $\boldsymbol{s}_a$ than $\boldsymbol{s}_n$.

3.3.3 Back to RAG: Retriever

Once the function $f(\cdot) = E(\cdot)$ to create semantic meaningful representations of sentences is defined, all the textual documents that form the KB must

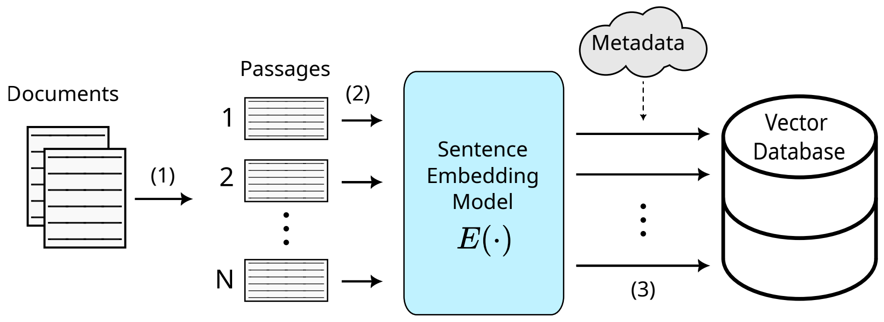


Figure 3.6: Retriever offline stage: (1) Chunking of the documents, (2) Compute sentence embeddings, and (3) store and index embeddings and metadata in a vector database.

be stored in a vector database for efficient retrieval: this stage is performed offline, i.e. before the actual usage of the RAG pipeline. An overview of the offline stage of the retriever is reported in Figure 3.6.

In the first step of this phase, (1) all the documents are split into small chunks, also called passages. Even if recent LLMs can handle long contexts, chunking remains an essential step in the pipeline because maintaining a constrained piece of information allows faster inference speed and quicker locates the original reference from which the generated content is extrapolated. Then (2) all the chunks are given in input to the sentence embedding model $E(\cdot)$, and (3) the embeddings are stored in a vector database. When storing the embeddings it is useful to add metadata to them, such as the title of the document, the section, and others. Metadata could be used for filtering: for example, in the case of one person knowing exactly in which document the information is looking for belongs, he can filter out all the irrelevant chunks reducing the noise of irrelevant retrieved passages. In addition to the common functionalities that a database offers, vector databases are built to improve the search and retrieval of vector embeddings by arranging them in a special data structure called vector index. To quickly search large datasets of vectors quickly, vector indexes rely on a class of algorithms called Approximate Nearest Neighbor (ANN) search, which given a query point finds a set of points close to the query. It is important to note that the speed of retrieval is paid in terms of the accuracy of the result, hence it is fundamental to make a trade-off based on the use case. Note that the creation of the index is done every time new passages are added to the database, but not during the RAG pipeline runtime. In this thesis project, we relied on the Qdrant vector database [32] that leverages the Hierarchical Navigable Small World (HNSW) [12] algorithm, which creates a hierarchical tree-like structure where each node of the tree represents a set of vectors and the edges the similarity between vectors.

The online stage of the retrieval starts with the user query to be encoded using the same embedding model $E(\cdot)$ that was used during the offline stage. Then, the query's embedding is compared by similarity with the vector index of the database to retrieve the top-k similar matches. Given two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, the most common metrics to compute the similarity are the cosine similarity 3.22 and the dot product 3.23.

$$S_C = \frac{\boldsymbol{u} \cdot \boldsymbol{v}}{||\boldsymbol{u}||||\boldsymbol{v}||} \tag{3.22}$$

$$\boldsymbol{u} \cdot \boldsymbol{v} = ||\boldsymbol{u}||||\boldsymbol{v}|| \cos\theta \tag{3.23}$$

The difference between the two is that cosine similarity considers only the direction of the vectors while the dot product varies based also on the magnitude. However, both are equally valid, and the choice should be made based on the similarity metric that was used during the training of the model.

3.4 Generation phase

In a RAG framework, the generator is responsible for creating the final output based on the most relevant information retrieved by the retriever, by synthesizing them into a coherent and contextually relevant response. This means that the final output should not be a mere collection of relevant information but should be presented in a way that addresses the requests of the user's query. Originally, in the RAG architecture proposed by [26], the generator used was a sequence-to-sequence model. The rationale is that the decoder must be strongly conditioned by the encoder output that includes the query and the retrieved documents. However, most recent works leverage state-of-the-art LLMs, both proprietary (e.g. GPT-4, Claude, etc.) and open-source (e.g. Llama 3, Mistral, etc.), which are typically autoregressive models. Since we tested the LLama and Mistral models as generators, we provide an overview of the main variations that they apply compared to the traditional transformer model described in Section 3.2.

3.4.1 The Llama family

Llama [48] is a family of decoder-only models that is trained using only publicly available datasets. Moreover, it introduces some variations with respect to the original transformer model.

First of all, instead of normalizing the output of each transformer sublayer, they add a pre-normalization of the input. Moreover, they moved from the LayerNorm normalizing function defined in Equation 3.15 to the RMSNorm function [21]. The RMSNorm for a generic vector $\boldsymbol{a} \in \mathbb{R}^n$ is:

$$\text{RMSNorm}(\boldsymbol{a}) = \frac{\boldsymbol{a}}{\text{RMS}(\boldsymbol{a})}\gamma, \quad \text{where } \text{RMS}(\boldsymbol{a}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2} \tag{3.24}$$

and γ is a parameter.

The second variation is the replacement of the ReLU non-linearity, defined in Equation 3.14 with the SwiGLU activation function [28] defined as:

$$\begin{gathered}\text{SwiGLU}(x, W, V, b, c) = \text{Swish}_\beta(xW + b) \otimes (xV + c),\\ \text{where } \text{Swish}_\beta(x) = x\sigma(\beta x),\end{gathered}$$

and σ is the sigmoid function.

Finally, they replaced the absolute positional embeddings with the rotatory positional embeddings [57]. This new approach of positional encoding consists of rotating the token embeddings in a multi-dimensional space with each dimension being rotated by an angle that is proportional to the token's position.

3.4.2 The Mistral family

Mistral 7B [44] is based on the Llama architecture, but introduces an important change to handle longer sequences more effectively at a reduced computational cost. This new feature is called Sliding Window Attention (SWA) and allows the model to process large inputs by focusing on smaller, overlapping chunks of the sequence, rather than attending to the entire sequence at once. In practice, each token can attend to at most W tokens from the previous layer, where W is the window size. Formally, the hidden state in position i of the layer k attends to all hidden states from the previous layer with position in $[i - W, i]$. Figure 3.7 shows the comparison between the traditional attention mechanism and the SWA with $W = 3$ and the recursive process of attending hidden states from previous layers.

Considering that the vanilla attention has a quadratic complexity with the sequence length $O(n^2)$ the SWA mechanism reduces the complexity to $O(n \cdot W)$.

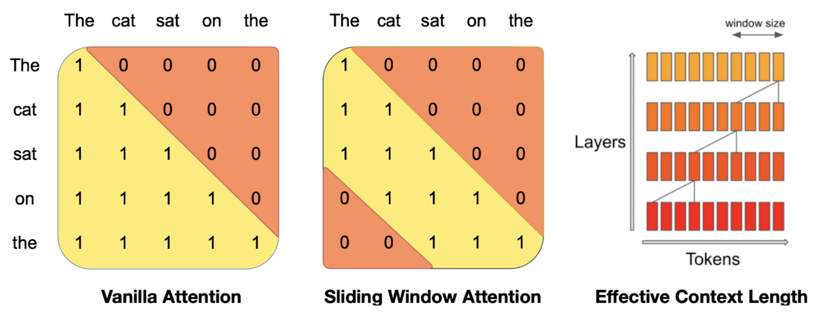


Figure 3.7: Comparison between the traditional attention mechanism and the SWA [44].

3.5 RAG systems

The RAG system presented above is the most simplistic one and is commonly known as Naive RAG. Even if it reduces common problems of LLMs, like hallucinations and outdated knowledge, it incurs other challenges to be addressed. Firstly, an ineffective retrieval phase leads to the selection of irrelevant documents affecting the entire pipeline. Then, even if the hallucination issue is reduced is not entirely solved, as the generator can produce irrelevant answers not supported by the retrieved content. Finally, the generator might repeat as it is the retrieved information without synthesizing, making the response impractical for the user. To address these issues more complex variations of the naive RAG have been introduced, by introducing new stages, like pre-retrieval and post-retrieval, or rearranging the pipeline [42].

3.5.1 Pre-retrieval stage

Most of the methods employed in the pre-retrieval stage involve modifying the user's query to improve the retrieval performance. Query transformation leverages the idea of modifying the original query of the user to formulate it in a clearer and more precise way and improve the retrieval performance. A simple approach is to follow a "rewrite-retrieve-read" paradigm, in which the user's query is, at first, given in input to an LLM, assigned with the rewrite role, and then the rewritten query is used for retrieval. When asking the LLM to rewrite the query for information retrieval it is fundamental to leverage on few shot prompting or train a model for this purpose, as done in [45]. Another transformation technique, called step-back prompting [51], consists of deriving a more general and easier question from the original query to retrieve a first set of relevant contexts and then combining them with the retrieved documents of the original query in the generation step. In the end, this method is composed of two steps: the abstraction, which is the generation of the step-back question and the retrieval of relevant documents about the generic question, and the reasoning, which is the final generation based on the information retrieved in the previous step. Another solution to handle complex queries is to decompose the original request into smaller problems [39]. This method is called least-to-most prompting and consists of two stages: the first one is the decomposition, which is done with a few shot prompting by providing some examples of decomposed queries, while the second is the subproblems solving. This second stage is again supported by a few shot prompting by providing some examples, and for each subproblem is appended the solution of the previous subproblem.

While previous methods modify directly the user's query, HyDE, presented in [34], leverages the query to create a fake document that helps in the retrieval of the relevant information. Specifically, HyDE generates a hypothetical document from a query using an instructed LLM; the fake

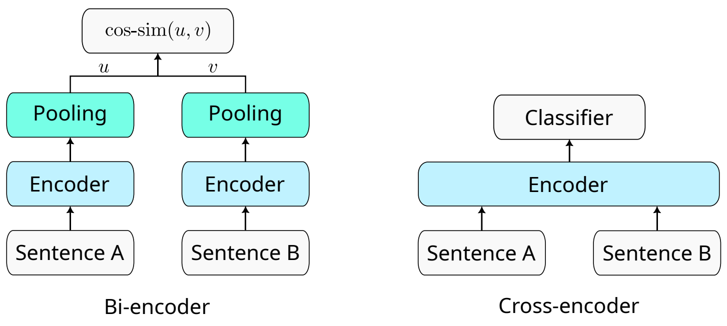


Figure 3.8: Comparison of the bi-encoder and cross-encoder architectures.

document does not have to contain the correct information, just to capture the relevance pattern to be the most similar to the real document.

3.5.2 Post-retrieval stage

The post-retrieval stage has the objective of improving the information that the generator uses to generate the final answer. This can be achieved by modifying the rank of the retrieved documents through reranking, which is a reordering of the retrieved chunks to improve the rank of the most relevant ones. A common and effective way to perform reranking is through cross-encoder models [20]. Compared to bi-encoder models, described in Section 3.3.2, in cross-encoders the two sentences are passed simultaneously to the same transformer network. A comparison of the two architectures is reported in Figure 3.8. Cross-encoders perform a full self-attention operation on both the two sentences and since they compare two inputs directly tend to have higher accuracy respect to bi-encoders. However, this increase in performance comes with a higher price in terms of computation efficiency. Indeed, they have to recompute the encoding for every new sentence. For example, in the case of information retrieval, where a query has to be compared with many documents, we have to recompute the encoding of the query for each document. Because of their high performance and slow computational time, they are usually used in IR systems to rerank a set of documents retrieved by a bi-encoder model.

3.5.3 Pipeline rearrangements

Another way to improve the RAG performance, rather than only adding new stages, is to rearrange the pipeline. Three possible rearrangements, iterative, recursive, and adaptive, are shown in Figure 3.9. An iterative solution has been proposed in [46]. With this approach retrieval and generation are repeated iteratively for T iterations. In particular, in addition to the classic retrieval-augmented generation paradigm, the architecture benefits from a generation-augmented retrieval stage. In this stage, retrieval is enhanced by appending the generated output of the previous iteration to

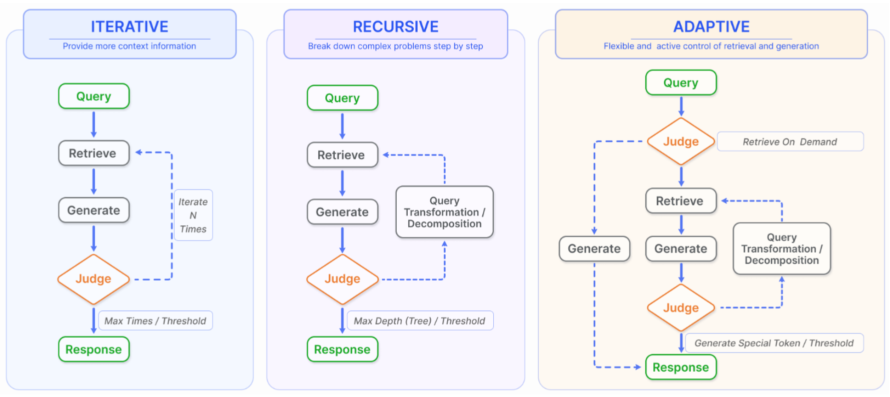

ITERATIVE
Provide more context information
Query
Retrieve
Generate
Judge
Iterate N Times
Max Times / Threshold
Response
RECURSIVE
Break down complex problems step by step
Query
Retrieve
Generate
Judge
Query Transformation / Decomposition
Max Depth (Tree) / Threshold
Response
ADAPTIVE
Flexible and active control of retrieval and generation
Query
Judge
Retrieve On Demand
Retrieve
Generate
Generate
Query Transformation / Decomposition
Judge
Generate Special Token / Threshold
Response

Figure 3.9: Three types of RAG pipelines (iterative, recursive, and adaptive) [42].

the query. The rationale behind this technique is to reduce the semantic gap between the query and the relevant document. Even if the generated content of the first iteration is not correct, it should be similar, in structure, to what might be needed to answer the question decreasing the semantic gap.

The recursive approach, as the iterative leverages a feedback loop to improve performance, involves modifying the search queries based on the results of the previous iteration. Hence, leverages some sort of query transformation to refine the user's query for better retrieval. An example of this methodology has been proposed in [38] where the author proposes a method to improve multi-step question-answering, based on an iterative approach of retrieval and reasoning through chain-of-thought prompting. After the first retrieval step, a language model performs reasoning with chain-of-thought on the question and the retrieved documents, and then the last generated sentence of the process is used as a query to retrieve more passages. The method ends when the set number of iterations is reached or when, during the chain-of-thought process, the string "The answer is:" is found.

Finally, the adaptive pipeline consists of automatically detecting when to leverage the additional information provided by the retriever or to rely only on the parametric memory of the generator. Self-RAG [41], besides determining if retrieval is useful given a query, critiques the retrieved documents to check if they are relevant, and the model's response for its usefulness and if it is supported by the context. This is done by generating special reflection tokens. In particular, the first step is to make a specific language model predict the reflection token to determine if continue on retrieval or move directly to the generation; then, if retrieval is needed, the documents are retrieved and answer generated; finally, the relevancy and the support score of the answer is predicted through others reflection tokens and the answer ranked based on the tokens values.

Chapter 4

Embedding model fine-tuning

In this chapter, we analyze the fine-tuning methods used in this work to improve the performance of the sentence embedding model in the retrieval stage. In particular, we describe supervised and self-supervised approaches and finally, we show how different tasks can be combined in an MTL setting.

4.1 Supervised

When fine-tuning with a supervised approach it is crucial to keep in mind the final role of the model. In our scenario, we need to boost the performance of IR on data in the space domain. In other words, we have to retrieve the relevant passage given a query. In practice, if we consider the vector representations of the relevant passage $p_+ \in \mathbb{R}^e$ and of the query $q \in \mathbb{R}^e$, where e is the embedding size, we want that q is closer to p_+ in the embedding space rather than to non-relevant passages p_-.

4.1.1 Multiple Negative Ranking Loss

The MNRL is an optimal choice since it minimizes the distance between an anchor and a positive sentence, and maximizes the distance between the anchor and the negative sentence. A great advantage of this method is that it needs to define only the positive example because the negative ones are the other samples in the batch. A batch is composed of B question passage pairs $(q_1, p_1), (q_2, p_2), \ldots, (q_B, p_B)$, where for (q_i, p_i), $p_i = p_+$, i.e.is a positive pair, and for (q_j, p_j) with $i \neq j$, $p_j = p_-$, i.e. are negative pairs. Hence, the loss for a single batch can be computed as:

$$L(q,p) = -\frac{1}{B}\sum_{i=1}^{B}\left[S(q_i,p_i) - \log\sum_{j=1}^{B} e^{S(q_i,p_j)}\right]. \tag{4.1}$$

Here, $S(\cdot)$ is the cosine similarity defined in Equation 3.22. In practice, the model is rewarded for reducing the angle between the embeddings of positive pairs and maximizing the angle between non-pair embeddings. A

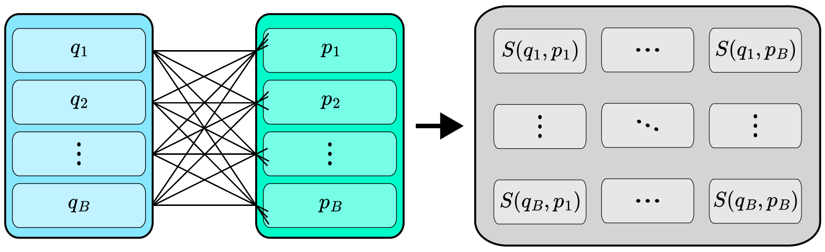


Figure 4.1: Computation of similarity in the MNRL. On the diagonal, there are the positive pairs that ideally should have the highest values.

graphical representation of the comparison between questions and passages is shown in Figure 4.1. Ideally, we would like to have a similarity equal to one on the diagonal and zero otherwise. Since in each batch, there are $B-1$ randomly sampled negative passages, a large batch size during training usually benefits the learning. Moreover, to make the training more effective it is important to ensure that no duplicate samples belong to the same batch to avoid redundancy.

4.2 Self-supervised

The advantage of self-supervised fine-tuning methods is that they do not require labeled training data. If on one side they allow training a model on domains where there are not labeled datasets, on the other side they perform rather poorly compared to a supervised approach. In the following, we analyze three self-supervised methods designed ad-hoc for training sentence embeddings.

4.2.1 TSDAE

The main idea behind TSDAE is to train sentence embeddings by adding a noise (e.g. deletion of words) to input sentences, encoding the damaged sentences into embeddings, and then reconstructing the embeddings into the original input. The training objective function of the reconstruction task for a single sample is formally defined as:

$$L(\boldsymbol{x}, \boldsymbol{h}) = \sum_{i=1}^{l} \log \frac{exp(h_i^\intercal x_i)}{\sum_{k=1}^{N} exp(h_i^\intercal x_k)}, \tag{4.2}$$

where $\boldsymbol{x}, \boldsymbol{h} \in \mathbb{R}^{l \times e}$ are respectively the word embedding of the input sentence with l tokens and the hidden state, and N is the vocabulary size.

Even if this encoder-decoder architecture seems similar to the one presented in Section 3.2, there is an important difference in the decoder. Indeed, if in the original transformer, the decoder takes into account all the encoder's output word embeddings, in the TSDAE case, the decoder has only access to the sentence representation produced by the encoder. This is the case because as shown in Figure 4.2 a pooling layer is introduced after the encoder. Formally, the output of the encoder's block in Equation 3.16 is defined as $\boldsymbol{H} \in \mathbb{R}^{1 \times s}$, where s is the sentence embedding size.

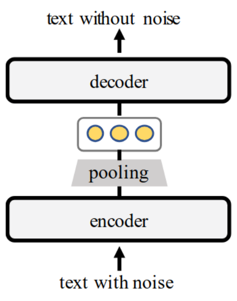


Figure 4.2: Architecture of TSDAE [33].

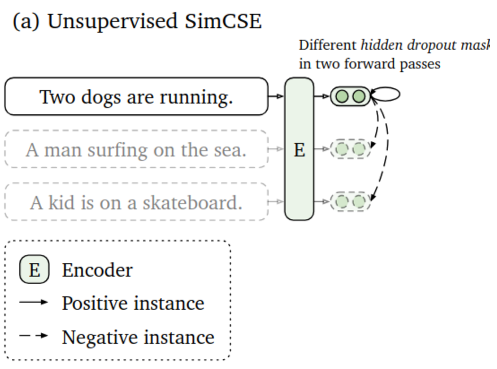

(a) Unsupervised SimCSE
Different hidden dropout mask in two forward passes
Two dogs are running.
A man surfing on the sea.
A kid is on a skateboard.
E
E Encoder
Positive instance
Negative instance

Figure 4.3: Unsupervised SimCSE architecture [31].

4.2.2 Unsupervised SimCSE

SimCSE is a contrastive learning framework to train sentence embeddings with an unsupervised approach. The objective of contrastive learning is to learn effective representations by pulling semantically close neighbors together and pushing apart non-neighbors. While the choice of negative samples is usually done in-batch, i.e. by considering non-neighbors all the samples in the same batch except for the only positive instance, the construction of neighbors is more complex. A common approach is to apply some sort of augmentation; for example, when working with images, positive instances can be created by flipping, rotating, or cropping the reference image. SimCSE leverages the standard dropout at the end of each sub-layer and to the sum of embeddings and the positional encodings in the transformer. In practice, they feed the same input to the encoder twice and get two embeddings with different dropout masks. A visual representation of the SimCSE architecture is depicted in Figure 4.3. Calling z and z' the two random dropout masks, the training objective function for SimCSE is:

$$L_i = -\log \frac{\exp(S(\boldsymbol{h}_i^{z_i}, \boldsymbol{h}_i^{z_i'})/\ \tau)}{\sum_{j=1}^{N} \exp(S(\boldsymbol{h}_i^{z_i}, \boldsymbol{h}_i^{z_i'})/\ \tau)}, \tag{4.3}$$

where $\boldsymbol{h}_i^z$ is the hidden state for sentence i with dropout mask z, $S(\cdot)$ is the cosine similarity defined in Equation 3.22, τ is a temperature hyperparameter, and N is the number of sentences in batch. The default dropout probability is set to $p_{drop} = 0.01$.

4.2.3 Contrastive Tension

The CT method leverages two independent models, initialized with identical weights. The two models are set to minimize the dot product of the sentence embeddings of different sentences and maximize the dot product for identical sentences. As in previous methods, the negative samples are the sentences in the same batch of the reference sentence, based on the assumption that

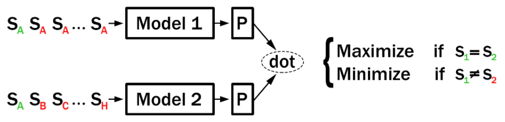


Figure 4.4: Architecture of CT [29].

two randomly selected sentences are very likely to be dissimilar. Figure 4.4 reporters a schema of the CT architecture. Given two sentences s_1 and s_2 and the two independent models $f_1(\cdot)$ and $f_2(\cdot)$, the CT objective is defined as:

$$z = f_1(s_1)^\intercal \cdot f_2(s_2)$$
$$L(z, s_1, s_2) = \begin{cases} -\log \sigma(z) & \text{if } s_1 = s_2 \\ -\log \sigma(1 - z) & \text{if } s_1 \neq s_2 \end{cases},$$

where $\sigma(\cdot)$ refers to the sigmoid function.

The tension referred to in the method's name is between the two models that have to obtain similar representations for identical sentences and at the same time distinguish their representation for different sentences. Doing so they act as anchors to each other remaining synchronized. After training the second model is usually used for inference.

4.3 Multi-task learning

Multi-task learning is a training paradigm in which a model is trained on multiple tasks simultaneously to learn them jointly. This approach can result in increased data efficiency, reducing the amount of data to train the network, a better generalization, and a reduction of overfitting through shared representations. Nevertheless, learning multiple tasks at once brings some challenges that if not handled properly make MTL ineffective. For example, trying to learn tasks that have conflicting needs yields performance improvements on one task and a performance degradation on the other. This phenomenon is referred as destructive interference or negative transfer. Moreover, MTL opens a lot of possible settings and configurations to combine multiple tasks, increasing the number of hyperparameters involved. An exisiting partition of MTL methods is based on how the parameters of the model are shared among multiple tasks. The first partition, called hard parameter sharing, involves those methods in which multiple weights are shared between multiple tasks so that each weight is trained to jointly minimize multiple loss functions. Secondly, in soft parameter sharing different tasks have individual task-specific models with separate weights. In this case, the distance between the model parameters of different tasks is added to the objective function [24].

4.3.1 Multi-Task Auxiliary Learning

Multi-Task Auxiliary Learning (MTAL) is a subset of MTL in which tasks are distinguished based on their importance regarding the model's final usage. The main or primary tasks are those that are required for the application output, while auxiliary tasks serve only to enrich the learning of the model and help it to better generalize. The objective of this approach is to optimize the generalization performance of a deep neural network on a main task $\mathcal{T}_m$, jointly trained with one or more auxiliary tasks $\mathcal{T}_a$. In general, this results in a multi-task loss that is a linear combination of task losses $\mathcal{L}_i$:

$$\mathcal{L}_{MTL}^{(j)} = \sum_{i=1}^{N_t} \lambda_i \mathcal{L}_i(\theta_s^{(j)}, \theta_i^{(j)}), \tag{4.4}$$

where $N_t = |\mathcal{T}_m| + |\mathcal{T}_a|$ is the total number of tasks, λ_i is the scaling factor, or weight, for task loss i, and θ_s, θ_i are the respectively the shared model parameters and the task-specific parameters of task i. During training at step j all parameters are updated through standard gradient descent:

$$\theta^{(j+1)} = \theta^{(j)} - \eta \nabla \mathcal{L}_{MTL}^{(j)}, \tag{4.5}$$

where η is the learning rate.

An interesting feature in this scenario is the choice of the task-specific weight λ_i. In the simplest configuration, the weights can be set to one for all tasks resulting in tasks' equivalently weighted. Otherwise, the weights can be set statically before training, for example by assigning smaller weights to the auxiliary tasks, in order to change the relative importance that tasks have during optimization.

Chapter 5

Experiments

This chapter is focused on the experimental part, starting from the definition of the data used as the KB and for training in Section 5.1, we move then to Section 5.2 to a description of the metrics used for the evaluation of the retriever and generator components. Finally, after reporting the experimental setup, the results of the experiments are shown and discussed. In particular, in Subsection 5.4.1, we study how fine-tuning on a supervised and MTL setting improves the performance compared to baseline models; then, in Subsection 5.4.2 we analyze the impact of the weight factor in MTL in order to find the value that lead to the best results; Subsection 5.4.3 objective is to determine how much the training set size affects the model's performance and to check if the amount of data collected for this work is enough for obtaining the best results; in Subsection 5.4.4 is discussed the variation of the generator's performance when embedding models with different capabilities are used, in order to determine how much the retriever component affects the generation phase; finally, Subsection 5.4.5, is a study on the usefulness of the entire RAG pipeline in a mission design during a CE session.

5.1 Knowledge Base and Datasets

In order to develop a RAG pipeline on space-related data, we collected 906 open-access scientific publications on the space domain. Even if the papers' topics are diverse the most recurrent ones are about earth observation missions, in-orbit demonstration and validation missions, Lunar and Deep Space missions, ground segment and telecommunication systems, and small satellites.

All the downloaded papers come in a PDF format and they must be converted into plain text in order to be preprocessed. To do that we leverage the Unstructured library [58], an open-source library that provides several tools for ingesting and pre-processing images and text documents, which uses Optical Character Recognition (OCR) techniques to convert text documents into machine-encoded text. The tool arranges the text into a JSON

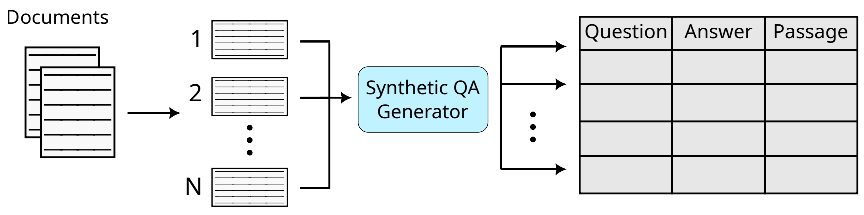


Figure 5.1: Workflow for synthetic QA generation.

Score	Score 1	Score 2	Score 3
Groundedness	761	3465	5900
Standalone	1071	3734	5375

Table 5.1: *Groundedness* and *standalone* scores provided by the Prometheus 2 judge model on synthetically generated samples.

format whose keys are the type of content (title, subtitle, narrative text, etc) of the file. After noticing some noise in the plain text we filtered out the pieces of text shorter than 10 characters. Even if by doing so the output is acceptable, in future steps it is important to spend more time on the data cleaning at this first step. Moreover, the data are now limited to PDF files only, but they can be easily extended to other textual sources.

After reading the PDF files, the next step is to divide the documents into smaller chunks. The Langchain library [55] provides a simple and efficient method to split the text recursively by using a set of chosen characters. In the end, we obtained 10126 chunks that compose the KB, where each chunk has a maximum size of 512 tokens, based on the BERT tokenizer.

For our purposes, two datasets have been created: one containing question, answer, and passage triples (*qap_triples*), which is needed for the pipeline evaluation and the fine-tuning supervised, and the second one simply made by sentences extracted from the papers (*sentences*), which is used for the self-supervised training. Driven by the findings of [49] who achieved good results by fine-tuning models on synthetic data only, we generated synthetic question-answer pairs from the passages of the KB, excluding the creation of an ad-hoc human-labeled dataset due to its heavy time and resource requirements. As the synthetic generator was used the Llama 8B model. Figure 5.1 shows the workflow for synthetic QA generation. Due to some noise and incorrect synthetic pairs, we exploited the Prometheus [54] LLM-as-a-judge model to score how much a question is *standalone*, i.e. if the question relies on additional information to be understood, and the *groundedness*, i.e. how much the question is grounded on the context. The judge model provides a score between 1 to 3, where the higher the score the better, and the triples which scored less than 2 were filtered, obtaining in the end 8348 valid samples. The distribution of the scores is depicted in table 5.1, while the prompts used for the synthetic data generator and the judge model can be found in the appendix A.1.1. To further make sure that the samples used for testing are reliable, they have been human-validated and modified when needed.

To obtain the *sentences* dataset we trivially extracted sentences by truncating at every period and and filtering those shorter than 25 characters. The final number of sentences that compose the dataset is 95730.

5.2 Evaluation Metrics

In this section are described the metrics used for the evaluation of the retriever and generator components.

5.2.1 Retriever Metrics

The retriever is evaluated with the following metrics:

Accuracy@k. Accuracy determines the average of the presence of a relevant document in the top k retrieved documents across the user's queries. In formulas can be defined as:

$$\text{accuracy}_k = \frac{1}{|Q|} \sum_{i=1}^{|Q|} rel_i \tag{5.1}$$

where Q are the user's queries and $rel_i \in 0, 1$ takes value 1 if at least a relevant document is present in the k retrieved results and 0 otherwise.

Normalized Discounted Cumulative Gain. NDCG is a ranking quality metric for information retrieval systems, based on the assumption that documents appearing earlier in a search result list, i.e. have higher ranks, are more relevant than documents that have lower ranks. The NDCG is determined by dividing the Discounted Cumulative Gain (DCG) by the ideal DCG (IDCG), which is the maximum possible DCG. Before describing the DCG, we introduce the concept of Cumulative Gain (CG) defined as:

$$\text{CG}_k = \sum_{i=1}^{k} g_i \tag{5.2}$$

where k is the cutoff point for looking at relevant items and g_i is the gain, or relevance score, of the result at position i. While the CG does not take into account the order of the search results, by adding a discount it is possible to penalize search results with low ranks. Then, the DCG is:

$$\text{DCG}_k = \sum_{i=1}^{k} \frac{g_i}{\log_2(i+1)} \tag{5.3}$$

In the formula above the logarithmic penalty makes dividing the gain by growing numbers as you move down the list of the search result list. To compute a score that is independent of the length of the result list, the DCG is normalized by the IDCG, which is the DCG computed when all relevant documents in the corpus are sorted by their relative relevance:

$$\text{IDCG}_k = \sum_{i=1}^{|rel_k|} \frac{g_i}{\log_2(i+1)} \tag{5.4}$$

where rel_k is the list of relevant documents in the corpus up to position k. Finally, the NDCG is calculated as:

$$\text{NDCG}_k = \frac{\text{DCG}_k}{\text{IDCG}_k} \tag{5.5}$$

Mean Reciprocal Rank. The MRR is, again, a ranking quality metric that considers the position of the first relevant item in the search result list. In particular, the MRR is the mean of reciprocal ranks, which are computed as the inverse of the position of the first relevant item, across all user's queries:

$$\text{MRR}_k = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i} \tag{5.6}$$

where Q are the user's queries and $rank_i$ is the position of the first relevant item for query i.

5.2.2 Generator Metrics

The automatic evaluation of generated text is still an open point in the research community. A good trade-off could be the combination of traditional n-gram overlap metrics, like the BLEU [3] and ROUGE [4] scores, semantic similarity scores, such as BERTScore [22] and SemScore [52], and LLM-as-a-judge [60, 54] evaluation.

BLEU. The Bilingual Evaluation Understudy (BLEU) algorithm was originally invented for evaluating machine translation tasks. In general, the score assesses how similar the candidate's text is to reference texts. The metric leverages a modified n-gram precision which is computed by counting the maximum number of times a word occurs in the reference, clipping the total count of each candidate word by its maximum reference count, adding them up, and dividing by the total number of candidate words. Formally:

$$p_n = \frac{\sum_{C \in Candidates} \sum_{g_n \in C} Count_{clip}(g_n)}{\sum_{C' \in Candidates} \sum_{g'_n \in C'} Count(g'_n)} \tag{5.7}$$

where g_n is an n-gram. In order to punish candidate strings that are to short a brevity penalty is introduced:

$$BP = \begin{cases} 1 & \text{if} \quad c > r \\ e^{(i-r/c)} & \text{if} \quad c <= r \end{cases} \tag{5.8}$$

being c the candidate's length and r the reference corpus size. Finally, the BLEU score is obtained:

$$\text{BLEU} = BP \ \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{5.9}$$

where w_n is a weight drawn by a chosen probability distribution.

ROUGE. Recall-Oriented Understudy for Gisting Evaluation (ROUGE) measures the quality of a generated text with respect to reference texts, by counting the number of overlapping units (n-grams, word sequences, word pairs) between the two. The most common ROUGE measures are ROUGE-N and ROUGE-L. We define as R the set of reference texts, as g_n the gram of length n, then:

$$\text{ROUGE-N} = \frac{\sum_{r \in R} \sum_{g_n \in r} C_{match}(g_n)}{\sum_{r \in R} \sum_{g_n \in r} C(g_n)} \tag{5.10}$$

where $C_{match}(g_n)$ is the number of n-grams overlapping in the candidate and reference texts, while Cg_n is the total number of n-grams in a piece of text. Instead, ROUGE-L is based on the longest common subsequence between the generated text and the reference, i.e. the longest sequence of words that is shared between both. It is calculated by simply change the numerator of equation 5.10 with the longest common subsequence, while the denominator is left unchanged for $n = 1$.

BERTScore. The metric computes the similarity score for each token, represented by an embedding, in the candidate sentence with each token in the reference sentence. Let us define $\boldsymbol{x}_i$ as the embedding of token x_i of the reference sentence, and $\boldsymbol{\hat{x}_i}$ as the embedding of token $\hat{x}_i$ of the candidate sentence. Then, the recall, precision, and F1 scores are computed as:

$$R_{BERT} = \frac{1}{|x|} \sum_{x_i \in x} \max_{\hat{x}_j \in \hat{x}} S_C(\boldsymbol{x}_i, \boldsymbol{\hat{x}}_j) \tag{5.11}$$

$$P_{BERT} = \frac{1}{|\hat{x}|} \sum_{x_j \in \hat{x}} \max_{x_i \in x} S_C(\boldsymbol{x}_i, \boldsymbol{\hat{x}}_j) \tag{5.12}$$

$$F_{BERT} = 2\frac{P_{BERT} \times R_{BERT}}{P_{BERT} + R_{BERT}} \tag{5.13}$$

where $S_C(\cdot)$ is the cosine similarity as defined in equation 3.22.

SemScore. This score is based on the comparison between the model output and a target response using semantic textual similarity. The model output and the target response are given in input to the same sentence transformer and then the cosine similarity is computed:

$$SemScore = S_c(\boldsymbol{x}, \boldsymbol{\hat{x}}) \tag{5.14}$$

where $\boldsymbol{x}$ is the target response and $\boldsymbol{\hat{x}}$ is the generated response.

METEOR. Metric for Evaluation of Translation with Explicit Ordering (METEOR) is used to evaluate the quality of machine translation outputs by comparing them with reference translations, and was designed to improve the BLEU score previously described. Initially, the unigram precision P and unigram recall (R) are computed as:

$$P = \frac{\text{Number of matched words in candidate}}{\text{Total number of words in candidate}} \tag{5.15}$$

$$R = \frac{\text{Number of matched words in candidate}}{\text{Total number of words in reference}} \tag{5.16}$$

and then are combined using the harmonic mean:

$$F_{mean} = \frac{P \times R}{\alpha P + (1-\alpha)R} \tag{5.17}$$

where in the original paper $\alpha = 0.9$. To take into account longer matches, a penalty is calculated as:

$$\text{Penalty} = \gamma(\frac{c}{m})^{\beta} \tag{5.18}$$

where c are groups of unigrams, called chunks, such that the unigrams in each chunk are in adjacent positions in the model's output and are also mapped to unigrams that are in adjacent positions in the reference text, m is the number of unigrams matched, and the tunable parameters are set to $\gamma = 0.5$ and $\beta = 3$. Finally, the METEOR score is computed as:

$$\text{METEOR} = F_{mean}(1 - \text{Penalty}) \tag{5.19}$$

LLM-as-a-judge. This unconventional evaluation approach, proposed in [60], makes use of an LLM to judge the responses of the tested model. The method is found to consistently agree with human preferences in addition to being capable of better evaluating open-ended tasks in comparison with traditional metrics. There are three proposed variations of the LLM-as-a-judge. The first one is the pairwise comparison, in which the judge is presented with a question and two answers, and tasked to determine which one is better. The second is a single answer grading, where the judge directly assigns a score to a single answer. Finally, in the reference-guided grading variation, the judge scores the answer in comparison to a reference solution. Even if this evaluation approach was initially proposed leveraging powerful proprietary models (e.g., GPT-4), in [54] the author proposes an open-source alternative called Prometheus 2, which has been used in this thesis work.

5.3 Experimental Setup

The sentence embedding models used for the experiments are the base and large versions of the BAAI General Embeddings (BGE) [50] family, and the medium and large versions of the *snowflake-arctic-embed* [56] models. The choice of these models is due to their high rank on the MTEB leaderboard [37] and their limited size with respect to top performers that are in the order of 7 billion of parameters. As generators, we tested the instruct versions of the Llama 3 8B and the Mistral 7B models.

Datasets. The two datasets used during the experiments are the *qap_triples* and *sentences* dataset. To totally separate the training and test sets, the test samples come from 20 papers that are not present in the training set. The resulting split in training, validation, and test set is reported in table 5.2.

Supervised fine-tuning. Each model has been fine-tuned with a supervised approach using the *qap_triples* dataset and the Multiple Negative Ranking Loss function. We trained the models for 20 epochs with a batch size of 32, learning rate $lr = 2e-05$, with weigh decay 0.001, a linear scheduler for the learning rate variation, no warmup steps, and using the AdamW optimizer. Moreover, we ensure that no duplicate samples are present in a batch. Leveraging Matryoshka Representation Learning [36], the models have been evaluated on the 512 and 256 embedding dimensions.

MTL fine-tuning. In the MTL approach, we made a linear combination of the self-supervised loss (TSDAE, SimCSE, CT) on the *sentences* dataset with the supervised loss. The models have been trained for 10 epochs with a batch size of 32, a learning rate $lr = 3e-06$, with weigh decay 0.001, a linear scheduler for the learning rate variation, no warmup steps, and the AdamW optimizer. Moreover, we multiplied the self-supervised loss by a scale factor and assessed the impact for different values in the range $[0.01, 1]$. As for the supervised setting, we leverage Matryoshka Representation Learning to evaluate the models on the 512 and 256 embedding dimensions.

Hardware componenets. All the experiments were done on two *Nvidia Tesla V100 SXM2* with 32 GB of VRAM. Computational resources provided by hpc@polito, which is a project of Academic Computing within

qap triples dataset		
Train	Validation	Test
7232	804	312
sentences dataset		
Train	Validation	Test
67011	9573	19146

Table 5.2: Split of train, validation, and test sets for the *qap_triples* and *sentences* datasets.

the Department of Control and Computer Engineering at the Politecnico di Torino (`http://www.hpc.polito.it`)

5.4 Results and discussion

In this section, we report and discuss the results of the fine-tuning of the embedding models described above, then we report some ablation studies, and finally, we show the results of a real case scenario during which the RAG pipeline has been tested on a mission design in Argotec.

5.4.1 Embedding models fine-tuning

Table 5.3 reports the results of the four tested models fine-tuned with a supervised and an MTL approach, using TSDAE as the self-supervised task, compared with the non-trained models' baseline, on different embedding dimensions.

In the MTL setting, the scaling factor on the self-supervised loss is set to $\lambda = 0.01$. We evaluated the accuracy at $k = 5$ because it is the number of passages that are given in input to the generator, while NDCG and MRR at $k = 10$ to have a general overview of the retriever performance. The MTL method outperforms all the others for all metrics, up to 3.86% increase, except for the *arctic-m* model where it achieves better results only on the NDCG@10 and MRR@10 metrics for the 768 embedding dimension. However, from the results is visible how fine-tuning the models with a supervised approach leads to a big improvement in the performance with respect to the baseline. Noteworthy, the *arctic-embed* model family, which performs poorly on our dataset, achieves the best results on all metrics after fine-tuning. Finally, the best-performing model on all metrics and embedding dimensions is the *arctic-l* fine-tuned with MTL. Considering the cost in terms of memory requirements due to the embedding dimension, a good trade-off between memory occupied and performance is the *arctic-l* model with embeddings truncated at 512, given the almost equivalent results with respect to the original size of 1024. In general, the degradation of performance between the original embedding dimension of the model and 512 is quite low, making it a good choice for all models.

In table 5.4, is reported the comparison between MTL fine-tuning using the TSDAE, SimCSE, and CT self-supervised tasks. While SimCSE is the lowest-performing method, TSDAE and CT obtain the best results for six cases each. In particular, the CT method performs better on the BGE family while TSDAE on the *snowflake-arctic-embed* family. Even if SimCSE achieves lower performance, its results are still comparable with other methods.

5.4.2 Impact of scale factor in MTL

Figure 5.2 depicts the variation of accuracy@5, NDCG@10, and MRR@10 for the fine-tuned *bge-base* model on the MTL setting with TSDAE as the self-supervised task, versus the scaling factor on the self-supervised loss. The main finding is that for all the scaling factor values, MTL brings an

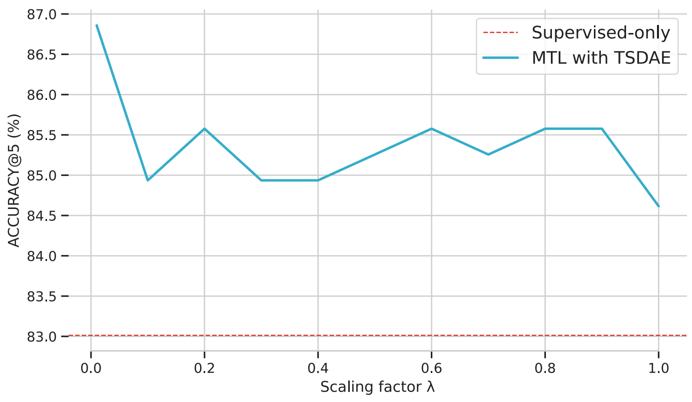
Model	Type	Accuracy@5			NDCG@10			MRR@10		
		Emb Dimension			Emb Dimension			Emb Dimension		
		-	512	256	-	512	256	-	512	256
bge-base	B	76.28	75.64	70.19	67.63	66.28	61.80	62.04	60.64	55.84
	S	83.01	83.01	82.05	73.87	73.86	71.90	68.07	68.47	66.35
	M	**86.87**	**85.58**	**84.30**	**74.44**	**74.53**	**73.40**	**68.83**	**68.85**	**68.58**
bge-large	B	84.94	81.73	79.17	75.09	72.09	68.45	70.29	66.37	63.14
	S	87.82	86.86	83.33	76.97	76.54	74.62	71.79	71.55	69.23
	M	**89.42**	**88.78**	**84.30**	**78.23**	**77.80**	**75.34**	**72.75**	**72.30**	**70.20**
arctic-m	B	15.38	14.10	09.62	12.99	12.18	08.92	11.34	10.21	07.65
	S	**91.99**	**90.71**	**88.46**	79.76	**79.54**	**77.52**	74.56	**74.43**	71.98
	M	90.06	88.78	87.50	**80.42**	78.63	77.22	**75.53**	73.74	**72.43**
arctic-l	B	13.78	13.14	12.18	12.42	11.63	11.47	10.57	10.19	09.33
	S	92.31	92.95	90.06	80.90	80.46	78.10	75.99	75.30	73.04
	M	**92.63**	**93.60**	**90.39**	**81.68**	**81.35**	**79.08**	**76.72**	**76.38**	**73.82**

Table 5.3: Comparison of IR metrics between baseline (B) models, and fine-tuned models, supervised-only (S) and MTL (M). Where the embedding dimension is not specified the model's original one is intended.

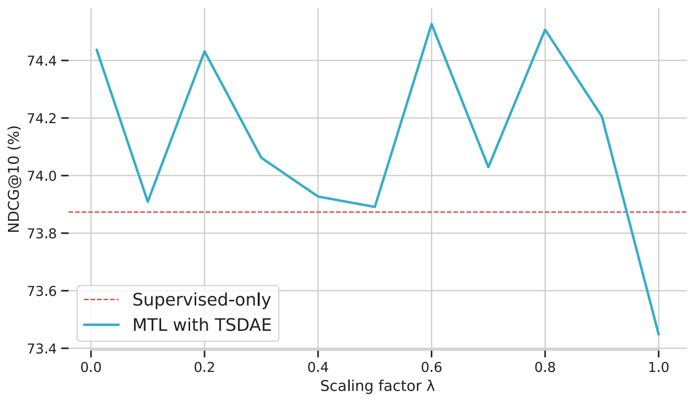
improvement with respect to supervised-only fine-tuning (except for the NDCG@10 with scaling factor $\lambda = 1$). In general, the lowest performance is obtained when $\lambda = 1$, i.e. when the supervised and self-supervised tasks have the same importance. The rationale could be that the self-supervised task, which is an additional task of embedding reconstruction, adds too much noise to the supervised task, which is the main one for information retrieval. Instead, when the self-supervised task is scaled it works as a regularization factor improving the overall performance.

5.4.3 Impact of training set size

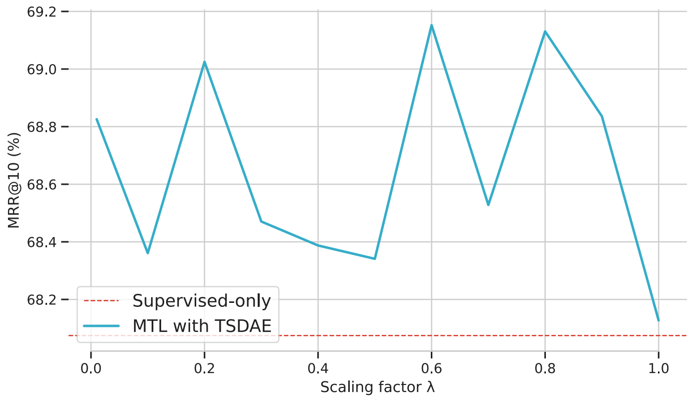
The lack of labeled data for specific domains is a huge problem. For this reason, we studied how the performance varies by changing the training set size for supervised fine-tuning. Since the drawn conclusions are valid for all models, only the tests on the *bge-base* model are reported. Considering the original training set dimension of 7232 samples, we tested the accuracy@5, NDCG@10, and MRR@10 metrics for 100, 1000, and 5000 training samples. The results are shown in Figure 5.3. Noteworthy, 100 samples are enough to improve all the metrics compared to the baseline, of at least 1%. Moreover, the three plots follow a trendline, in which the improvements increase quickly and then level out. However, the limitation of our maximum training size does not allow to see where the metrics reach convergence, meaning that a bigger training set could have further improved the performance of the model.



(a) Accuracy@5

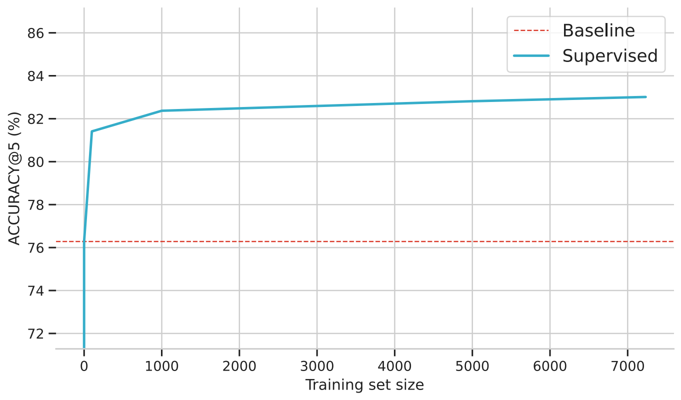


(b) NDCG@10

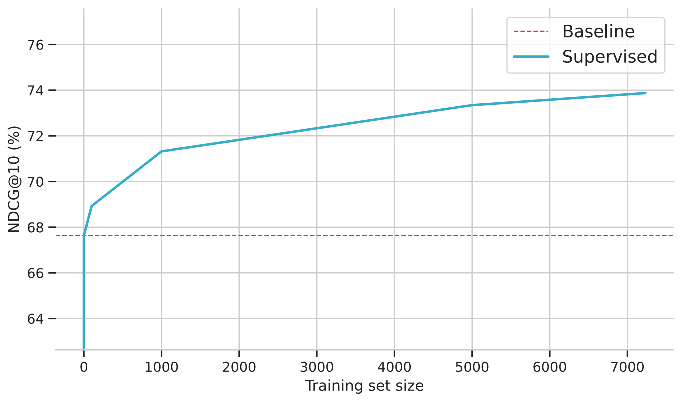


(c) MRR@10

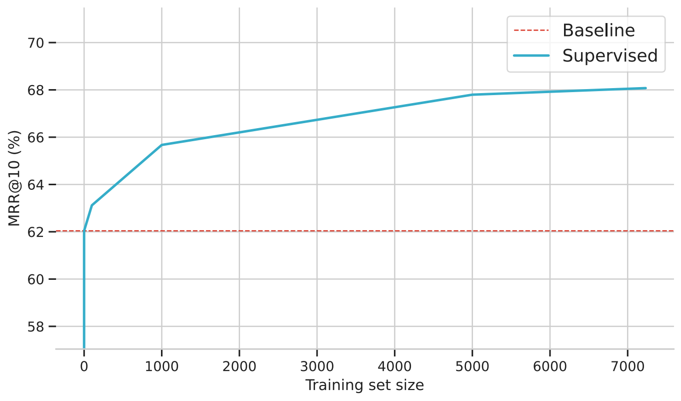
Figure 5.2: Performance variation for the *bge-base* model versus the scaling factor variation on the TSDAE task in MTL.



(a) Accuracy@5



(b) NDCG@10



(c) MRR@10

Figure 5.3: Performance variation for the *bge-base* model versus different training set sizes for supervised fine-tuning.

Model	Method	Accuracy@5	NDCG@10	MRR@10
bge-base	TSDAE	**86.87**	74.44	68.83
	SimCSE	85.90	74.44	68.97
	CT	85.90	**75.16**	**69.41**
bge-large	TSDAE	89.42	78.23	72.75
	SimCSE	87.82	75.32	69.70
	CT	**90.06**	**78.52**	**73.30**
arctic-m	TSDAE	90.06	**80.42**	**75.53**
	SimCSE	90.39	79.20	73.68
	CT	**91.35**	79.39	73.80
arctic-l	TSDAE	**92.63**	**81.68**	**76.72**
	SimCSE	92.95	80.92	75.98
	CT	91.99	80.44	75.41

Table 5.4: Comparison between the MTL setting with different self-supervised tasks (TSDAE, SimCSE, CT).

5.4.4 Generator evaluation

To assess the impact of the retriever performance on the generator, we evaluated the generated answer using the metrics described in Section 5.2.2. Regarding the usage of the Prometheus 2 model as a judging LLM, we define three metrics: the correctness, to assess if the generated answer is correct according to the reference answer, the faithfulness, to determine how much the model's output is supported by the context of the retrieved documents, and the relevancy, to evaluate how well the generated response aligns with the intent of the initial query. The prompts given in input to the judge model can be found in Appendix A.1.3. As a first study, we compared the Llama 3 and Mistral performance using our best embedding model (the *snowflake-arctic-embed-l* fine-tuned on the MTL setting with TSDAE) and its baseline. The results are reported in Table 5.5. For both generators the usage of our model lead to a great improvement of the metrics, confirming the importance of a good retrieval stage for the entire RAG pipeline performance. However, since the baseline of the tested embedding model performs quite poorly according to Table 5.3, we run the experiment using the *bge-base* model fine-tuned on MTL setting and its baseline since the difference between the two is lower. For this second test, the results are shown in Table 5.6. Again, the fine-tuned version leads to an improvement, even if smaller compared to the previous case, demonstrating the importance of further fine-tuning on domain-specific data in an already high-performing model.

5.4.5 Real case scenario on mission design

To assess the effectiveness of the RAG pipeline, it has been tested with questions that have been already answered during a real mission design session.

Generator	Emb. Model	Corr.	Faith.	Relev.	BERTScore	SemScore	Rouge-L	BLEU	METEOR
Llama3 8B	arctic-l	2.099	2.370	2.308	56.69	73.22	18.00	6.27	26.73
	arctic-l-space	**4.048**	**3.468**	**4.112**	**69.33**	**82.81**	**39.02**	**14.65**	**58.26**
Mistral 7B	arctic-l	2.318	2.599	2.683	56.61	73.58	16.13	4.49	26.13
	arctic-l-space	**4.061**	**3.388**	**4.218**	**66.88**	**81.70**	**31.94**	**11.17**	**51.19**

Table 5.5: Impact of the best embedding model (*snowflake-arctic-embed-l*) on generators performance.

Generator	Emb. Model	Corr.	Faith.	Relev.	BERTScore	SemScore	Rouge-L	BLEU	METEOR
Llama3 8B	bge-base	3.865	3.385	3.987	66.84	81.53	34.72	12.71	52.61
	bge-base-space	**3.974**	**3.503**	**4.035**	**67.98**	**82.22**	**37.32**	**13.37**	**56.13**
Mistral 7B	bge-base	3.856	3.304	4.196	64.73	80.33	28.01	9.45	46.71
	bge-base-space	**4.141**	**3.321**	**4.279**	**66.14**	**81.22**	**29.77**	**10.37**	**48.70**

Table 5.6: Impact of the *bge-base* embedding model on generators performance.

During this process, 16 questions have been given in input to the model. The three participants in the pilot study assessed the quality of the responses, providing scores between 1 and 5 based on four criteria [53]: coverage, how much the model's response covers all the user's requests, providing all necessary information; consistency, how much the generated answer is aligned to the reference context, i.e. the information provided in the document; correctness, the output states affirmations that are verifiable and within the scope of user's information request; clarity, if the response is expressed in a clear and understandable way. Table 5.7 reports the average scores on the 16 questions, while table 5.8 shows a pair of samples of generated answers and references. In appendix A.2, table A.1 shows the mean average scores per question. The results demonstrate that the pipeline effectively satisfies users' requests by providing on average complete and correct answers with clarity. Outputs with low scores were primarily due to missed retrieval of the correct passages; however, this issue can be mitigated by rephrasing the question and adding more details.

Coverage	Consistency	Correctness	Clarity
4.21	3.98	4.29	4.44

Table 5.7: Human evaluation on four criteria (coverage, consistency, correctness, and clarity, of the responses generated by the RAG pipeline.

Question	Generated answer	Reference
Which is the NEA scout mission requirement on the solar phase angle?	According to the provided documents, the NEA Scout mission requirement on the solar phase angle is to provide lighting conditions conducive to imaging, with a solar phase angle of less than 50 degrees.	Less than 50 degrees
What type of solar sail did the IKAROS mission have and what is its area?	According to the provided documents, the IKAROS mission had a square-shaped solar sail with a side length of 14 meters, which corresponds to an area of approximately 196 m2.	IKAROS sail is a disk type sail, not rigid with an area of 196 m2

Table 5.8: Samples of questions, generated answers, and reference answers from the 16 questions of the mission design study.

Chapter 6

Conclusion

To summarize, we employed a RAG framework to aid system engineers to efficiently retrieve information regarding past missions during CE sessions. In particular, this kind of architecture allows to interrogate external documents that might go beyond the parametric knowledge of LLMs, like for example company internal reports. Moreover, due to privacy constraints, we rely on open-source models both for the generator and retriever side. However, existing models are usually trained on general-purpose data and they lack on domain-specific knowledge. For this reason, we decided to invest in the retriever improvement through fine-tuning on space-related data. We mainly focused on supervised and MTL training and for this reason, we needed a dataset for training and evaluation. Unfortunately, collecting a dataset supervised by human operators is extremely expensive and time-consuming; hence, we discarded this option in Argotec. So we moved to a quicker alternative to generate synthetic data exploiting LLMs capabilities. However, to increase the reliability of the results, as synthetic data could include errors, we human-validated the test set.

The results of fine-tuning on four models show strong improvements on both the supervised and MTL settings. In particular, the best results are obtained with MTL and by scaling, with a fixed weight factor, the self-supervised loss component. We tested MTL with three different self-supervised methods and found that the best performance is achieved with TSDAE and CT. Moreover, we tried to assess the impact of the retriever component, before and after fine-tuning, on the generator, showing an improvement for the two tested models. Finally, we tried to assess the usefulness of the RAG pipeline within a real session of mission design. To do that, the engineers involved in the case study evaluated the model's response on four criteria. These results are promising but preliminary, and further assessment is required to fully understand the tool's potential in the context of CE.

In the future, a crucial improvement would be to invest in explainability to increase the trust toward the pipeline. Trivially, reporting the documents attended by the generator to generate the response will avoid searching through documentation to check if the model's answer is correct. Then, the tool is now limited to IR, but in the next steps, we would like a VA that helps in the exploration of new ideas in mission design. Moreover, in this work, we leveraged the naive RAG architecture, but, as described in Section 3.5, many other enhancements of the pipeline, besides fine-tuning the retriever, can be performed.

To conclude, this thesis work is the first attempt for Argotec to integrate NLP techniques inside the company, and the promising results achieved by this first proof of concept are pushing the company to invest more in this sector.

Appendix A

Appendix title

A.1 Prompts

In this section are reported the prompts used during the experiments.

A.1.1 Prompts for QA generation

The prompt used for the generation of synthetic question-answer pairs is:

Your task is to write a question and an answer given a context. Your question should be answerable by retrieving information from the context. Your question should encourage reasoned and discoursive answers, not a simple retrieval of information from the context. Your answer should be motivated and concise, but including only the information asked with question. This means that your question MUST NOT mention something like "according to the passage" or "context".
Provide your answer as follows:
Output:
Question: (your question)
Answer: (your answer to the question)
Now here is the context.
Context: {context}
Output:::

The prompt used by the Prometheus 2 judge model to assess how much a question is standalone is:

Task Description:
An instruction (might include an Input inside it), a response to evaluate, and a score rubric representing a evaluation criteria are given.
1. Write a detailed feedback that assess the quality of the response strictly based on the given score rubric, not evaluating in general.
2. After writing a feedback, write a score that is an integer between 1 and

5. You should refer to the score rubric.
3. The output format should look as follows: "(write a feedback for criteria) [RESULT] (an integer number between 1 and 5)"
4. Please do not generate any other opening, closing, and explanations.
The instruction to evaluate: Your task is to evaluate how the given question does not rely on additional information to be understood.
Question: {question}
Score Rubrics: [Does the question make sense by itself?]
Score 1: If the question does not make sense by itself.
Score 2: If the question is partially understendable by itself.
Score 3: If the question make totally sense by itself.
Feedback:

while to assess if the question is grounded on the context is:

Task Description: An instruction (might include an Input inside it), a response to evaluate, and a score rubric representing a evaluation criteria are given.
1. Write a detailed feedback that assess the quality of the response strictly based on the given score rubric, not evaluating in general.
2. After writing a feedback, write a score that is an integer between 1 and 5. You should refer to the score rubric.
3. The output format should look as follows: "(write a feedback for criteria) [RESULT] (an integer number between 1 and 5)"
4. Please do not generate any other opening, closing, and explanations.
The instruction to evaluate: Your task is to evaluate how well one can answer the generated question unambiguously with the given context.
Generated question: {question}
Context: {context}
Score Rubrics: [Is the question answerable with the given context?]
Score 1: If the question is not answerable with the given context.
Score 2: If the question is partially answerable with the given context.
Score 3: If the question is totally answerable with the given context.
Feedback:

A.1.2 Prompt for the RAG generator

Follows the prompt of the RAG generator used in the experiments.

Using the information contained in the context, give a comprehensive answer to the question.
Respond only to the question asked, response should be concise and relevant to the question.

If the answer cannot be deduced from the context, do not give an answer.
Context:
{*context*}
Now here is the question you need to answer.
Question: {*question*}

A.1.3 Prompts to assess RAG generator

Here are reported the prompts used to define the metrics of faithfulness, relevancy, and correctness for the Prometheus 2 model.
Task Description:
An instruction (might include an Input inside it), a context, an answer to evaluate, and a score rubric representing a evaluation criteria are given.
1. Write a detailed feedback that assesses the quality of the response strictly based on the given score rubric, not evaluating in general.
2. After writing a feedback, write a score that is either 1 or 2 or 3 or 4 or 5. You should refer to the score rubric.
3. The output format should look as follows: "Feedback: (write a feedback for criteria) [RESULT] (1 or 2 or 3 or 4 or 5)"
4. Please do not generate any other opening, closing, and explanations.
5. Only evaluate on common things between generated answer and context.
The instruction to evaluate: Your task is to evaluate if the generated answer is supported by context.
Generated answer: {*generated_answer*}
Context: {*context*}
Score Rubrics: [Does the generated answer reflects the information contained in the context?]
Score 1: If the generated answer does not contain any information contained in the context.
Score 2: If the generated answer contain information contained in the context but explained with low clarity.
Score 3: If the generated answer contain information contained in the context but in a inconsistent way.
Score 4: If the generated answer contain information contatined in the context explained with clarity.
Score 5: If the generated answer contain information contained in the context explaind with clarity and consistency.
Feedback:

Task Description:
An instruction (might include an Input inside it), a query, a response to evaluate, and a score rubric representing a evaluation criteria are given.
1. Write a detailed feedback that assesses the quality of the response strictly

based on the given score rubric, not evaluating in general.
2. After writing a feedback, write a score that is either 1 or 2 or 3 or 4 or 5. You should refer to the score rubric.
3. The output format should look as follows: "Feedback: (write a feedback for criteria) [RESULT] (1 or 2 or 3 or 4 or 5)"
4. Please do not generate any other opening, closing, and explanations.
5. Only evaluate on common things between generated answer and context.
The instruction to evaluate: Your task is to evaluate how well the generated response aligns with the intent and content of the initial query.
Query: {question}
Generated answer: {generated_answer}
Score Rubrics: [Is the generated answer relevant to the intent and content of the initial query?]
Score 1: If the generated answer is not relevant to the user query.
Score 2: If the generated answer is relevant to the user query but do not meet the query's specific requirements.
Score 3: If the generated answer is relevant to the user query and meet only a subset of query's specific requirements.
Score 4: If the generated answer is relevant to the user query and meet the query's specific requirements but add not requested information.
Score 5: If the generated answer is relevant to the user query and meet all the query's specific requirements without adding not requested information.
Feedback:

Task Description:
An instruction (might include an Input inside it), a query, a response to evaluate, a reference answer that gets a score of 5, and a score rubric representing a evaluation criteria are given.
1. Write a detailed feedback that assesses the quality of the response strictly based on the given score rubric, not evaluating in general.
2. After writing a feedback, write a score that is either 1 or 2 or 3 or 4 or 5. You should refer to the score rubric.
3. The output format should look as follows: "Feedback: (write a feedback for criteria) [RESULT] (1 or 2 or 3 or 4 or 5)"
4. Please do not generate any other opening, closing, and explanations.
5. Only evaluate on common things between generated answer and reference answer.
Don't evaluate on things which are present in reference answer but not in generated answer.
The instruction to evaluate: Your task is to evaluate the generated answer and reference answer for the query: {question}
Generate answer to evaluate: {generated_answer}
Reference Answer (Score 5): {reference_answer}
Score Rubrics: [Is the generated answer relevant to the user query and reference answer?]

Question	Mean Coverage	Mean Consistency	Mean Correctness	Mean Clarity
1	5	5	5	4,67
2	5	4,67	5	5
3	4,33	5	5	5
4	4,33	1,33	1,33	5
5	4,33	4,33	4,33	4,33
6	5	5	5	3,66
7	4	4	4,67	3
8	4,33	4,67	5	3,67
9	3,37	3,37	3,67	4,67
10	2	1	2,33	4,67
11	4	4,67	4,67	4,37
12	4,37	4	4,33	4
13	3,33	2,67	3,67	4,67
14	5	5	5	5
15	4,33	4,67	4,67	4,67
16	4,67	4,33	5	4,67

Table A.1: Mean scores for each question in the real case scenario test on the coverage, consistency, correctness, and clarity metrics.

Score 1: If the generated answer is not relevant to the user query and reference answer.
Score 2: If the generated answer is according to reference answer but not relevant to user query.
Score 3: If the generated answer is relevant to the user query and reference answer but contains mistakes.
Score 4: If the generated answer is relevant to the user query and has the exact same metrics as the reference answer, but it is not as concise.
Score 5: If the generated answer is relevant to the user query and fully correct according to the reference answer.
Feedback:

A.2 Results on real case scenario

Table A.1 reports the mean scores of the 16 questions of the real case scenario test on the coverage, consistency, correctness, and clarity metrics.

Bibliography

[1] 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*. 1997, pp. 16–24.

[2] Massimo Bandecchi, B Melton, and Franco Ongaro. "Concurrent engineering applied to space mission assessment and design". In: *ESA bulletin* 99.Journal Article (1999).

[3] Kishore Papineni et al. "Bleu: a method for automatic evaluation of machine translation". In: *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 2002, pp. 311–318.

[4] Chin-Yew Lin. "Rouge: A package for automatic evaluation of summaries". In: *Text summarization branches out*. 2004, pp. 74–81.

[5] Stephen Robertson, Hugo Zaragoza, et al. "The probabilistic relevance framework: BM25 and beyond". In: *Foundations and Trends® in Information Retrieval* 3.4 (2009), pp. 333–389.

[6] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[7] Kyunghyun Cho et al. "On the properties of neural machine translation: Encoder-decoder approaches". In: *arXiv preprint arXiv:1409.1259* (2014).

[8] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems* 27 (2014).

[9] Matthew Henderson et al. "Efficient natural language response suggestion for smart reply". In: *arXiv preprint arXiv:1705.00652* (2017).

[10] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[11] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[12] Yu A Malkov and Dmitry A Yashunin. “Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs”. In: *IEEE transactions on pattern analysis and machine intelligence* 42.4 (2018), pp. 824–836.

[13] F Murdaca et al. “Artificial intelligence for early design of space missions in support of concurrent engineering sessions”. In: *8th International Systems & Concurrent Engineering for Space Applications Conference*. 2018.

[14] Alec Radford et al. “Improving language understanding by generative pre-training”. In: (2018).

[15] Zhenzhong Lan et al. “Albert: A lite bert for self-supervised learning of language representations”. In: *arXiv preprint arXiv:1909.11942* (2019).

[16] Mike Lewis et al. “Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension”. In: *arXiv preprint arXiv:1910.13461* (2019).

[17] Yinhan Liu et al. “Roberta: A robustly optimized bert pretraining approach”. In: *arXiv preprint arXiv:1907.11692* (2019).

[18] Antoni Viros i Martin and Daniel Selva. “Daphne: A virtual assistant for designing earth observation distributed spacecraft missions”. In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 13 (2019), pp. 30–48.

[19] Alec Radford et al. “Language models are unsupervised multitask learners”. In: *OpenAI blog* 1.8 (2019), p. 9.

[20] Nils Reimers and Iryna Gurevych. “Sentence-bert: Sentence embeddings using siamese bert-networks”. In: *arXiv preprint arXiv:1908.10084* (2019).

[21] Biao Zhang and Rico Sennrich. “Root mean square layer normalization”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[22] Tianyi Zhang et al. “Bertscore: Evaluating text generation with bert”. In: *arXiv preprint arXiv:1904.09675* (2019).

[23] Tom Brown et al. “Language models are few-shot learners”. In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[24] Michael Crawshaw. “Multi-task learning with deep neural networks: A survey”. In: *arXiv preprint arXiv:2009.09796* (2020).

[25] Pengcheng He et al. “Deberta: Decoding-enhanced bert with disentangled attention”. In: *arXiv preprint arXiv:2006.03654* (2020).

[26] Patrick Lewis et al. "Retrieval-augmented generation for knowledge-intensive nlp tasks". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 9459–9474.

[27] Colin Raffel et al. "Exploring the limits of transfer learning with a unified text-to-text transformer". In: *Journal of machine learning research* 21.140 (2020), pp. 1–67.

[28] Noam Shazeer. "Glu variants improve transformer". In: *arXiv preprint arXiv:2002.05202* (2020).

[29] Fredrik Carlsson et al. "Semantic re-tuning with contrastive tension". In: *International Conference on Learning Representations, 2021*. 2021.

[30] Shijie Chen, Yu Zhang, and Qiang Yang. "Multi-task learning in natural language processing: An overview". In: *ACM Computing Surveys* (2021).

[31] Tianyu Gao, Xingcheng Yao, and Danqi Chen. "Simcse: Simple contrastive learning of sentence embeddings". In: *arXiv preprint arXiv:2104.08821* (2021).

[32] Qdrant. *Qdrant Vector Database*. 2021. URL: https://qdrant.tech/.

[33] Kexin Wang, Nils Reimers, and Iryna Gurevych. "Tsdae: Using transformer-based sequential denoising auto-encoder for unsupervised sentence embedding learning". In: *arXiv preprint arXiv:2104.06979* (2021).

[34] Luyu Gao et al. "Precise zero-shot dense retrieval without relevance labels". In: *arXiv preprint arXiv:2212.10496* (2022).

[35] Andres Garcia-Silva et al. "Spaceqa: Answering questions about the design of space missions and space craft concepts". In: *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2022, pp. 3306–3311.

[36] Aditya Kusupati et al. "Matryoshka representation learning". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 30233–30249.

[37] Niklas Muennighoff et al. "MTEB: Massive Text Embedding Benchmark". In: *arXiv preprint arXiv:2210.07316* (2022). DOI: 10.48550/ARXIV.2210.07316. URL: https://arxiv.org/abs/2210.07316.

[38] Harsh Trivedi et al. "Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions". In: *arXiv preprint arXiv:2212.10509* (2022).

[39] Denny Zhou et al. "Least-to-most prompting enables complex reasoning in large language models". In: *arXiv preprint arXiv:2205.10625* (2022).

[40] Josh Achiam et al. "Gpt-4 technical report". In: *arXiv preprint arXiv:2303.08774* (2023).

[41] Akari Asai et al. “Self-rag: Learning to retrieve, generate, and critique through self-reflection”. In: *arXiv preprint arXiv:2310.11511* (2023).

[42] Yunfan Gao et al. “Retrieval-augmented generation for large language models: A survey”. In: *arXiv preprint arXiv:2312.10997* (2023).

[43] Ziwei Ji et al. “Survey of hallucination in natural language generation”. In: *ACM Computing Surveys* 55.12 (2023), pp. 1–38.

[44] Albert Q Jiang et al. “Mistral 7B”. In: *arXiv preprint arXiv:2310.06825* (2023).

[45] Xinbei Ma et al. “Query rewriting for retrieval-augmented large language models”. In: *arXiv preprint arXiv:2305.14283* (2023).

[46] Zhihong Shao et al. “Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy”. In: *arXiv preprint arXiv:2305.15294* (2023).

[47] Shamane Siriwardhana et al. “Improving the domain adaptation of retrieval augmented generation (RAG) models for open domain question answering”. In: *Transactions of the Association for Computational Linguistics* 11 (2023), pp. 1–17.

[48] Hugo Touvron et al. “Llama: Open and efficient foundation language models”. In: *arXiv preprint arXiv:2302.13971* (2023).

[49] Liang Wang et al. “Improving text embeddings with large language models”. In: *arXiv preprint arXiv:2401.00368* (2023).

[50] Shitao Xiao et al. *C-Pack: Packaged Resources To Advance General Chinese Embedding.* 2023. arXiv: `2309.07597 [cs.CL]`.

[51] Huaixiu Steven Zheng et al. “Take a step back: Evoking reasoning via abstraction in large language models”. In: *arXiv preprint arXiv:2310.06117* (2023).

[52] Ansar Aynetdinov and Alan Akbik. “SemScore: Automated Evaluation of Instruction-Tuned LLMs based on Semantic Textual Similarity”. In: *arXiv preprint arXiv:2401.17072* (2024).

[53] Lukas Gienapp et al. “Evaluating generative ad hoc information retrieval”. In: *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval.* 2024, pp. 1916–1929.

[54] Seungone Kim et al. “Prometheus 2: An open source language model specialized in evaluating other language models”. In: *arXiv preprint arXiv:2405.01535* (2024).

[55] Langchain. *Langchain library.* 2024. URL: `https://www.langchain.com/`.

[56] Luke Merrick et al. “Arctic-Embed: Scalable, Efficient, and Accurate Text Embedding Models”. In: *arXiv preprint arXiv:2405.05374* (2024).

[57] Jianlin Su et al. “Roformer: Enhanced transformer with rotary position embedding”. In: *Neurocomputing* 568 (2024), p. 127063.

[58] Unstructured. *Unstructured library.* 2024. URL: `https://unstructured.io/`.

[59] Penghao Zhao et al. “Retrieval-augmented generation for ai-generated content: A survey”. In: *arXiv preprint arXiv:2402.19473* (2024).

[60] Lianmin Zheng et al. “Judging llm-as-a-judge with mt-bench and chatbot arena”. In: *Advances in Neural Information Processing Systems* 36 (2024).