University of Padova

Department of Physics and Astronomy "Galileo Galilei"

Master Thesis in Physics of Data

Domain-Specific Cross-Lingual RAG Applications: Information Retrieval Fine Tuning with Synthetic Data Generation

Supervisor
Prof. Jacopo Pazzini
University of Padova

Co-supervisor
PhD Candidate Stefano Campese
University of Trento

Master Candidate
Lorenzo Barbiero

Student ID
2082142

Academic Year
2024-2025

"I think, therefore I am"
— René Descartes

Abstract

In the context of Retrieval-Augmented Generation (RAG) systems, the development of high-performance retrieval algorithms for effective query-document matching is essential to answer the user's queries, especially with complex queries spanning multiple documents. This thesis proposes a novel framework leveraging generative capabilities of large language models to generate domain-specific, synthetic query-document pairs in a cross-lingual setting, where queries are created in multiple languages (including high and low-resource languages) and documents are primarily in Italian. The synthetic data generated is used to fine-tune transformer based dense sentence encoders, enabling more effective information retrieval.

The proposed approach introduces a query generation pipeline to enhance retrieval efficiency while preserving semantic integrity. Through comprehensive experiments, the framework demonstrates that fine-tuning smaller, task-specific models using high-quality synthetic data can outperform state-of-the-art retrieval solutions in terms of accuracy and computational efficiency (up to $+7.5\%$ in retrieval MAP@10 and $+4.4\%$ in question answering accuracy).

This work highlights the potential of cross-lingual synthetic data generation as a cost-effective and scalable solution for improving domain-specific information retrieval in RAG applications, especially in scenarios involving multilingual queries and limited annotated data. This approach not only improves efficiency but also addresses critical security concerns. By enabling the evaluation of sensitive data to be conducted locally on company machines, risks associated with data leaks are significantly mitigated, resulting in enhanced compliance with data protection regulations.

Contents

Listing of figures

Listing of tables

1 Introduction

In recent years, the development of highly capable Large Language Models (LLMs) has fundamentally transformed the way we interact with text-based artificial intelligence systems. These models' ability to understand the deep semantic meaning of text, as well as their capacity for contextual awareness and memory of previous conversations, has made LLM-based chatbots exceptionally efficient and reliable. Retrieval-Augmented Generation [1] (RAG) is a framework designed to enhance the capabilities of language models by incorporating external knowledge through retrieval-based methods. In a RAG system, when a query is asked, it is first routed to an information retrieval component. This component uses various techniques to locate and return relevant documents that can inform the query, which are then incorporated into the generation prompt as context, allowing the LLM to generate an answer. By combining retrieval and generation, RAG systems improve the model's ability to handle complex queries and produce more accurate, contextually relevant responses, particularly in Question Answering (QA) tasks. RAG is especially valuable in knowledge-intensive, rapidly evolving, or domain-specific areas where the data available to the LLM during training may not include the necessary information to answer specific questions.

The three basic building blocks of RAG systems are shown in Figure 1.1:
Retrieval: The retrieval component is responsible for fetching relevant information from an external knowledge base or database. This is essential for providing the model with context or facts that might not be directly encoded within its training data or model parameters. The input query is encoded into a large dimensional vector representation, in which small euclidean distance is associated with close semantic meaning. From there, the most relevant documents from the database are selected and retrieved based on a similarity measure in the vector space. The retrieval step allows the system to access real-world information and address out-of-scope queries, ensuring that the generated output is informed by up-to-date and domain-specific knowledge.

Augmentation: The augmentation step involves using the retrieved documents to enhance or "augment" the

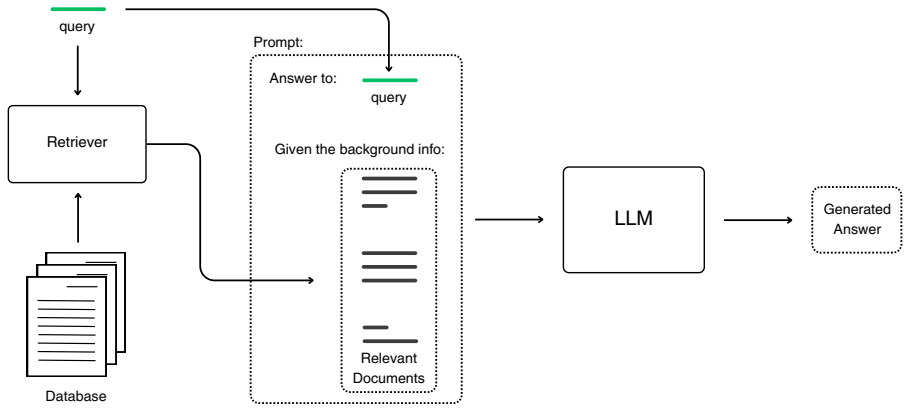


Figure 1.1: RAG Schema: The query (green) is augmented with relevant documents from a database using an information retriever. The joint information is used by the LLM to provide the final answer.

input that is fed to the generative model. The retrieved information is concatenated with the original query or prompt, providing richer context. The goal is to integrate the relevant information effectively, ensuring that it contributes to generating a coherent and informed response.

Generation: The generation component is the core of a RAG system, where a large language model produces a response based on the augmented input (query + retrieved documents). This component is typically powered by a transformer-based generative model (generally a LLM) that has been fine-tuned for natural language generation tasks. The generative model processes the input, which now includes both the original query and the retrieved documents, and generates a response that is contextually informed by the augmented data. The goal of the generation step is to produce an output that is not only coherent and fluent, which LLMs excel at, but also grounded in the external information provided by the retrieval and augmentation phases.

In this dissertation, we propose a novel end-to-end pipeline that significantly enhances the performance of RAG systems in cross-lingual, domain-specific use case scenarios through the utilization of a custom synthetic dataset. By leveraging the natural language processing capabilities of LLMs, we generate synthetic but human-like queries based on private company documents that can't be part of the LLM training data. These queries are subsequently employed to fine-tune the information retrieval component of the RAG system. Our proposed pipeline shows superior performance compared to state-of-the-art models in end-to-end RAG evaluation.

This thesis will be structured as follows: chapter 2 is a theoretical introduction and background to fully understand the techniques and models used during this research. Chapter 3 will describe the data generation and validation process, the following information retrievers fine-tuning and their evaluation in retrieval and Question-Answering tasks. Chapter 4 contains the final assesments on the results, the conclusions and ideas for further research.

2

Natural Language Processing Techniques

This chapter covers the essential theoretical background for understanding Retrieval-Augmented Generation systems. We start by introducing Natural Language Processing (NLP), a research field that combines linguistics, computer science, and artificial intelligence (AI) to help machines understand, interpret, and generate human language. We'll look at how NLP has evolved, from early rule-based systems to statistical methods, and finally to the modern neural network approaches at the heart of this research. Next, we focus into Machine Learning (ML), a core component of AI that powers today's NLP techniques. We'll explore the fundamentals of ML and focus on artificial neural networks (ANNs), which are key to understanding more complex models discussed later. We'll also touch on neural network architectures particularly effective in NLP. Among these, Sequence-to-Sequence (Seq2Seq) models stand out for tasks like machine translation, summarization, and question answering. We'll cover Recurrent Neural Networks (RNNs), Long Short-Term Memory networks (LSTMs)[2], and Gated Recurrent Units (GRUs)[3], highlighting their strengths, limitations, and uses in NLP.

At the core of this chapter is an in-depth look at the Transformer [4] architecture. Building on earlier Seq2Seq models, Transformers have become the foundation of modern language models. We'll explore how their key components drive superior performance across various NLP tasks, from text generation to embedding, with a special focus on Large Language Models. We'll also briefly review how large language models are deployed in real-world applications, giving insight into their industrial use. To wrap up, we focus on two key areas in modern NLP systems: text generation augmentation and query-document retrieval. In text generation, we introduce prompt engineering—how carefully crafted prompts help guide models to produce more accurate, context-aware outputs. This section sheds light on how to tune and leverage the power of large language models. Finally, we discuss retrieval methods, which are crucial for RAG systems. From traditional to advanced neural approaches, we examine how these techniques improve query-document matching and integrate with generative models to create more informative, accurate responses.

2.1 Natural Language Processing

Natural Language Processing is a subfield of AI and computational linguistics that focuses on the interaction between computers and human (natural) languages. The goal of NLP is to enable machines to understand, interpret, generate, and respond to human language in a way that is both meaningful and contextually appropriate, bridging the gap between human communication and computer understanding. Nowadays, NLP is well integrated in everyday use with various applications such as Search Engines (Google, Bing, ...), Text Generation (ChatGPT, Bing Chat), Text to Image generators (DALL-E, Midjourney), Vocal assistants (Siri, Alexa, ...), automated fake news detection and online content moderation.

The need for NLP arises from the fact that human processing power is insufficient to handle the vast amount of text generated daily by human activities, making automatic processing essential. Additionally, language analysis and processing are inherently complex due to a variety of factors:

- Word Vocabulary: The vocabulary employed in everyday use or even in specialized fields encompasses a huge variety of distinct terms. On top of that, new terms are constantly added as the language evolves. NLP needs to work with a large number of entities and adapt to new ones.
- Word frequency Distributions: Word frequency in a language is highly skewed in favor of few high frequency words and a long tail of rare words. In a random text corpus word frequency is approximated by a Power Law (Zipf's Law [5]) with

  $$\text{word frequency} \propto \frac{1}{\text{word rank}}$$

  Where the rank is obtained from ordering terms in a text corpus from the most to the least common. NLP needs to stay consistent across orders of magnitude in word frequency.
- Language Structure: The message in a text is not only defined by the words themselves, which in turn can also have multiple meanings, but by the whole combination of meaning, sentence structure and context. NPL needs to develop a global perspective on text.

This makes NLP tasks particularly challenging which have, in turn, led to the development of cutting edge architectures that have revolutionized the AI field as a whole.

2.1.1 NLP History

To understand modern state of the art NLP it is useful to focus briefly on the history of NLP and the various ways in which language has been processed. We will discuss the strengths and weaknesses associated with different frameworks and asses the current state of NLP. At its core, NLP history can be divided into three major periods:

Symbolic NLP (**1950s − 1990s**): Symbolic NLP, also known as rule-based NLP, was the first major era in the development of NLP systems. During this period, complex linguistic rules and symbolic representations were used to process language. These rules, often written by highly specialized linguists, were based on formal grammar and symbolic representations of language elements (words, phrases, sentences). The primary goal was

to represent the syntactic and semantic structure of sentences using logic and rules for parsing, understanding, and generating language in a manner that machines could manipulate. This era often used formal languages like context-free grammars to parse sentences. Relying on rigid, human-crafted rules made the models ill suited for scalability, flexibility, and complexity. Writing extensive rules for all possible sentence structures was both time-consuming and error-prone for large datasets, since language is inherently ambiguous. The main applications of this era include basic machine translation on limited datasets and entity-name recognition.

Statistical NLP($\mathbf{1990s - early\ 2010s}$): Statistical NLP represented the shift from hand-crafted rules to data-driven approaches. The rise of computational power and large corpora of text data led to the adoption of statistical methods that could automatically learn patterns and relationships in language from vast amounts of text data. Statistical NLP could handle larger datasets and models could be trained on data directly, making it easier to improve their performance. In addition, statistical models could better handle ambiguities in language by relying on context and probabilities rather than strict rules. The most important applications in this era are Hidden Markov Models, Statistical Machine Translation and N-Gram language models. Despite the improvements over the previous approach, statistical NLP still relied on high quality annotated data and models that required expertise and time to create and optimize. On top of that, long range dependencies and context remained a challenge.

Neural NLP($\mathbf{mid\ 2010s - today}$): The Neural NLP era represents the most transformative phase in NLP, driven by deep learning techniques and neural networks. This era has been marked by significant breakthroughs in the performance of NLP models, particularly due to advancements in neural architectures like recurrent neural networks, long short-term memory networks and, especially, transformer based models. Deep learning models are the backbone of recent AI breakthroughs, displaying outstanding performance on a wide range of NLP tasks, including machine translation, question answering, and sentiment analysis. This stems from the increased ability to generalize and understand deep correlations between different pieces of text using contextual embeddings, where words are represented in high-dimensional vector spaces in which semantically similar words are closer together. The two paramount models in Neural NLP so far have been BERT[6] for text embedding and GPT-3[7] for text generation. Neural models are typically trained in an end-to-end manner, where the model learns directly from a large corpus of raw text data to perform a task (e.g., translation, classification) without requiring manual feature engineering or dataset annotation. This has also led to the development of transfer learning, in which large pre-trained models are fine-tuned on specific tasks, thus allowing models to leverage vast amounts of general knowledge before being tailored to specific uses. However, reliance on large datasets also represents a drawback. Training large models on vast amounts of data is energy intensive, which is both expensive and environmentally impactful. On top, the usage of large datasets has the possibility to reflect unwanted biases in the models, leading to ethical concerns regarding fairness.

2.2 Machine Learning

Machine Learning is a branch of artificial intelligence that focuses on the development of algorithms and statistical models that allow computers to perform specific tasks and generalize to unseen data without explicit programming. Instead of receiving step-by-step instructions for every possible scenario, machine learning systems learn from data itself. At their core, ML algorithms are extremely powerful and expressive correlation finders, that is, they learn from the data by finding correlations and exploit this knowledge to make predictions and carry out tasks, often producing results that are more accurate than traditional data modeling methods. Machine learning techniques are applied across various domains, from natural language processing and computer vision to recommendation systems, fraud detection, and autonomous vehicles.

2.2.1 The Learning Framework

In the first section, a set of general and foundational concepts for any machine learning problem and algorithm will be presented.

Supervised, Unsupervised, Reinforcement Learning

Traditional machine learning can be divided into three main branches:

- **Supervised Learning**: In this approach, the model is trained on a labeled dataset, where each data point is associated with a desired output, called label. The goal is to learn a function that correctly maps inputs to outputs, allowing the model to make accurate predictions on new, unseen data. Supervised learning excels in well-defined tasks with clear success criteria. However, it requires large amounts of labeled data, which can be difficult, expensive, time-consuming, or impossible to obtain.
- **Unsupervised learning** operates on unlabeled data; its objective is to discover hidden structures or intrinsic patterns in the data without the guidance of predefined outputs. Unsupervised learning is powerful for data exploration and insight discovery, but it can be challenging to evaluate since there is no "ground truth" to compare results against.
- **Reinforcement learning** is based on interaction with an environment. In this approach, an agent learns to make decisions by performing actions in an environment to maximize a cumulative reward. RL is particularly powerful in sequential decision-making scenarios and has achieved notable successes in fields such as game playing and robotic control. However, it can be computationally intensive and requires careful design of the reward system.

In the wake of new models and data retrieval techniques some additional hybrid learning frameworks have been implemented in order to leverage the strengths of both supervised and unsupervised learning.

- **Self-Supervised Learning**: Self-supervised learning (SSL) is a type of machine learning where a model learns to predict parts of the input data from other parts, without needing manually labeled data. The model generates its own supervision by creating labels from the data itself. SSL avoids the costly process of labeling data, which is a major limitation in many machine learning tasks. In SSL, the system typically creates a pretext task (e.g., predicting the next word in a sentence, or filling in missing pixels in an image) to learn useful features from the data. After this, the model can be fine-tuned on a downstream task with labeled data.

– **Semi-Supervised Learning**: Semi-supervised learning lies between supervised and unsupervised learning. It uses a small amount of labeled data and a large amount of unlabeled data to train a model. The model leverages the labeled data for guidance and the unlabeled data to generalize. The model is trained on both labeled and unlabeled data: the labeled data helps the model understand the correct output for specific examples, while the unlabeled data allows the model to recognize patterns and structures in the data.

Evaluation Metrics for NLP

Evaluation metrics are numerical quantities used to assess the effectiveness of the model and its ability to generalize. Without introducing the more basic metrics such as accuracy, precision and recall we will only focus on retrieval and ranking specific metrics. In a retrieval task the goal is to, given a question (query), find the most relevant documents within a database. For simplicity we can consider each document either relevant (R) to the query or non-relevant (N).

– **Precision**: precision up until result k ($P@k$) is defined as

$$P@k = \frac{1}{k}\sum_{i=1}^{k} \text{rel}(x_i) \qquad \text{where} \qquad \text{rel}(x) = \begin{cases} 1 & \text{if } x \text{ is relevant } R \\ 0 & \text{if } x \text{ is not relevant } N \end{cases}$$

It represents the percentage of relevant results up until result k

Example - Precision

Suppose that the search results at $k = 5$ are

$$\text{res } @5 = [R, R, N, R, N] \quad \rightarrow \quad P@5 = \frac{1}{5}(1 + 1 + 0 + 1 + 0) = 0.6$$

– **AP**: Average Precision ($AP@k$) is defined as

$$AP@k = \frac{1}{\text{GTP}}\sum_{i=1}^{k} P@i \ \text{rel}(x_i)$$

GTP (Ground Truth Positives) are the number of possible correct answers. The advantage over average precision is the fact that now ranking order matters and, for $k_2 > k_1 \implies AP@k_2 > AP@k_1$ which was not a property of $P@k$.

Example - Average Precision

Suppose to have $GTP = 4$ and two search results

$$\text{res}_1 \text{@}5 = [R, R, N, R, N] \qquad \text{res}_2 \text{@}5 = [N, N, R, R, R]$$

The first search result is clearly better than the second but the precision metric is not able to capture this, in fact

$$P_1\text{@}5 = P_2\text{@}5 = 0.6$$

However, computing average precisions

$AP_1\text{@}5 = \frac{1}{4}\sum_{i=1}^{5} P_1\text{@}i \ \text{rel}(x_i) = \frac{1}{4}(1 \cdot 1 + 1 \cdot 1 + \frac{2}{3} \cdot 0 + \frac{3}{4} \cdot 1 + \frac{3}{5} \cdot 0) = 0.686$
$AP_2\text{@}5 = 0.358$

Shows that $AP_1\text{@}5 > AP_2\text{@}5$.

– **MAP**: Mean Average Precision at k ($MAP@k$) is defined as

$$MAP\text{@}k = \frac{1}{N}\sum_{i=1}^{N} AP\text{@}k$$

Where N is the sample size

Example - Mean Average Precision

For the two previous results

$$MAP\text{@}5 = \frac{1}{2}(AP_1\text{@}5 + AP_2\text{@}5) = 0.522$$

– **NDCG**: Normalized discounted cumulative gain, similarly to MAP, applies a penalty term (discount) to the retrieval position. First, the Discounted Cumulative Gain (DCG) is defined

$$DCG\text{@}k = \sum_{i=1}^{k} \frac{\text{rel}_i}{\log_2(i+1)}$$

DCG can then be normalized by computing the Ideal DGC, that is, the DGC in the case of perfect retrieval

$$NDCG\text{@}k = \frac{DCG\text{@}k}{IDCG\text{@}k}$$

Example - NDCG

Suppose to have $GTP = 4$ and search result

$$\text{res}\,@5 = [R, R, N, R, N]$$

then

$$DCG@5 = \frac{1}{\log_2 2} + \frac{1}{\log_2 3} + \frac{1}{\log_2 5} = 2.06$$

$$IDCG@5 = \frac{1}{\log_2 2} + \frac{1}{\log_2 3} + \frac{1}{\log_2 4} + \frac{1}{\log_2 5} = 2.56$$

and so

$$NDCG@k = 0.805$$

These metrics are going to be extremely relevant in assessing the quality of the research presented in the next chapter.

2.3 Neural Networks

Neural networks are a fundamental component of modern machine learning, serving as the foundation for deep learning algorithms that have revolutionized fields such as computer vision, robotics and, most importantly Natural Language Processing. Inspired by the structure and function of biological neural networks in the human brain, artificial neural networks are powerful computational models capable of learning complex patterns and relationships in data.

Structure

A neuron in an artificial neural network is a computational unit that receives inputs, processes them, and produces an output. The output can be passed to other neurons or can serve as as the final prediction of the network. Each neuron performs the following operations:

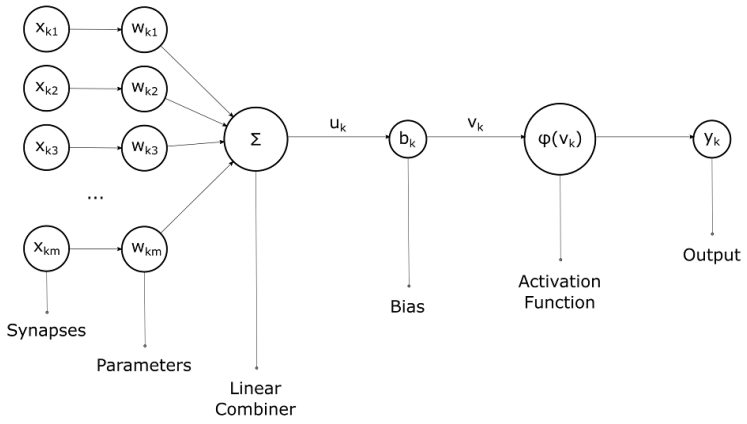


Figure 2.1: Example of neuron computation: The neuron receives the signal from the previous layer neurons x_i, each multiplied by their corresponding weights w_i, and summed together with the bias. The result is then passed through the activation function φ and serves as the output.

- Inputs: Neurons receive inputs, often represented as numerical values, either from the data or from other neurons in the previous layer.
- Weights: Each input is multiplied by a weight that indicates the strength of the connection between neurons. Weights determine how much influence each input has on the neuron's output.
- Bias: A bias term is added to the weighted sum of the inputs. It allows the neuron to adjust the output independently of the inputs, improving the flexibility of the model.
- Activation Function: After summing the weighted inputs and adding the bias, the result is passed through an activation function. The activation function introduces non-linearity into the network, allowing it to learn and model complex relationships within data. Without activation functions, a neural network would essentially behave like a complex linear regression model, unable to capture intricate patterns and structures.

The equation for a single neuron can be represented as:

$$\text{out} = \varphi\big(\sum(\vec{x}_i \cdot \vec{w}_i) + b\big)$$

Where:

1. x_i are the inputs
2. w_i are the corresponding weights
3. b is the bias
4. φ is the activation function

Neurons are tipically organized into layers, with each layer processing information before passing it to the next layer. Layers are divided into three main types:

- Input Layer: The first layer in the neural network, it receives the raw data (features) and feeds it into the next layer. The input layer doesn't perform any computation but is responsible for passing the data to the first hidden layer.
- Hidden Layers: These are the intermediate layers where the actual computation happens. Each neuron in a hidden layer receives inputs from the previous layer, processes them, and passes the output to the next layer. The hidden layers allow the network to learn complex patterns and representations. A neural network can have one or multiple hidden layers.
- Output Layer: The output layer produces the final result or prediction. For example, in a classification problem, the output layer may have one neuron per class, and the output could represent the probabilities of different classes. In regression tasks, it could have a single neuron representing the predicted value.

The architecture of a neural network, including the number of layers and neurons in each layer, is a critical factor in its performance. More layers and neurons allow the network to learn more complex representations, but this also increases the computational complexity and risk of over-fitting.

Training procedure and Backpropagation

Training a neural network means adjusting the parameters (weights and biases) of the network to minimize the error in its predictions. This process is done through a combination of forward propagation and backpropagation, iteratively refining the model to perform well on unseen data. The three steps involved in training a Neural Network are:

- Initialization: Before training starts, the neural network's weights and biases are typically initialized with random values. This is done to break symmetry, so that neurons don't learn the same features.
- Forward Propagation: data is passed through the network from the input layer to the output layer. The output of the final layer is then compared to the actual target (truth) to compute the loss, remember how a lower loss means that the network is making better predictions. The goal of training is to minimize this loss function.
- Backpropagation: Backpropagation is the core algorithm for adjusting the weights and biases in a neural network. It involves calculating the gradient (the partial derivatives) of the loss function with respect to each weight and bias in the network. These gradients indicate how much the weights need to change to reduce the error. The process is called Gradient Descent

Gradient Descent works by using the chain rule of calculus to propagate errors backward from the output layer to the input layer. This allows the network to adjust weights in the direction that minimizes the error. The weight update rule in gradient descent is given by:

$$w_i \to w_i - \eta \frac{\partial L}{\partial w_i}$$

Where:

- w_i is the weight
- η is the learning rate (a hyperparameter that controls the step size)
- $\frac{\partial L}{\partial w_i}$ is the gradient of the loss function with respect to the weight w_i.

The biases are updated in a similar fashion to weights, using the computed gradients. The learning rate η controls the step size in the direction of the gradient, that is, how sensitive it is to new updates. If the learning rate is too high, the model might overshoot the optimal solution and fail to converge while, if it is too low, it may take too long to converge. Backpropagation and weight updates are repeated over many iterations until, ideally, the network's loss reaches an acceptable minimum, meaning the model has learned from the data and is making accurate predictions.

Deep Neural Networks

A Deep Neural Network (DNN) is a neural network with a large number of hidden layers between the input and output layers. The main advantage of DNNs over simpler architectures is the increased representation power, which allows them to learn complex data and detect relevant features from raw data, eliminating the need for hand-crafted feature engineering. This makes them extremely effective for traditionally difficult tasks such as Computer Vision and natural language processing. However, training deep neural networks presents a difficult challenge for two main reasons:

- Vanishing and Exploding Gradients: during backpropagation, the gradients of the loss function can become extremely small (vanishing) or large (exploding) as they are propagated backward through layers. This makes training deep networks difficult because the weights in the earlier layers can receive either very small updates or excessively large updates.
- Overfitting: DNNs with a large number of parameters are prone to overfitting, where the network learns the noise in the training data instead of generalizing to unseen data.

Residual Networks (ResNets) [8], are a specific type of deep neural network that alleviates some of the challenges associated with training very deep networks. The main innovation of ResNets is the residual block, which introduces skip connections that allow the output of a layer to bypass one or more layers. A residual block is a building block of ResNet and can be thought of as a shortcut around one or more layers. The key idea is to add the input of the block to its output:

$$\text{output} = \text{activation}(W \cdot \text{input} + b) + \text{input}$$

The input to the block is added directly to the output of the block before applying the activation function. This creates a "residual" connection that helps the model learn the the difference between the input and the output of the block, instead of the whole transformation from input to output. Skip connections solve most of the problems which affected standard DNNs:

- Mitigating Vanishing Gradients: By adding skip connections, the gradients can flow more easily during backpropagation, reducing the vanishing gradient problem. This allows for the training of much deeper networks.
- Improved Training: Skip connections make it easier for the network to learn identity functions (i.e., functions where the output is close to the input). When these identity functions are learned, the network can skip unnecessary transformations.
- Faster Convergence: Empirical results show that ResNets converge faster than traditional deep neural networks because of the effective gradient flow and reduced risk of overfitting.

It is easier for the network to learn the residual (or difference) between the input and output than it is to directly learn the output. This simplifies the training of deep networks. In a very deep network, if a certain block does not learn anything useful, the skip connection allows the network to retain the original input in the output, effectively skipping the block's computation. ResNets are particularly well-suited for tasks that involve very deep architectures and are at the core of the recent breakthroughs in NLP.

2.3.1 Seq2Seq Networks

Sequence-to-Sequence models are a class of neural network architectures designed for tasks where both input and output are sequences, this is a common need in NLP tasks such as machine translation, text summarization and text generation, in which the input and output texts are usually of different length. The fundamental idea behind Seq2Seq models is to take a sequence of elements as input and transform it into another sequence, potentially of a different length, with the network learning how to map from the input space to the output space. The model architecture generally consists of two primary components: an encoder and a decoder. The encoder processes the input sequence and compresses it into a fixed-size context vector (or hidden state), which is then passed to the decoder, generating the output sequence. Over the years different architectures have been developed for both encoding and decoding and in this chapter the most important ones will be discussed.

Architectures

RNN:

Recurrent Neural Networks (RNNs) were the original architecture used for sequence-based tasks. Unlike traditional ANNs, RNNs process data through multiple steps, allowing them to capture dependencies between elements in the sequence (Figure 2.2). They operate by maintaining a hidden state that is updated one step at a time as the network processes each element of the sequence, updating their hidden state based on the current input and the previous hidden state (Figure 2.3). RNNs share weights across time steps, which helps them generalize better and process sequences of varying lengths. In the simplest form, a RNN update rule can be thought as

$$f_\theta : (x_t, h_t) \rightarrow (y_t, h_{t+1})$$

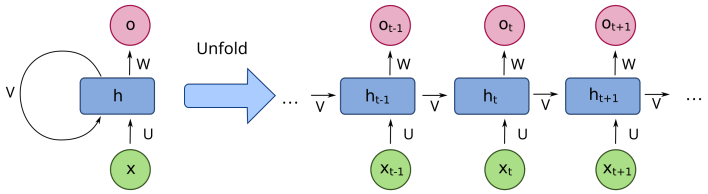


Figure 2.2: General idea behind recurrent architectures: an input sequence is processed one element at the time updating the internal state of the network

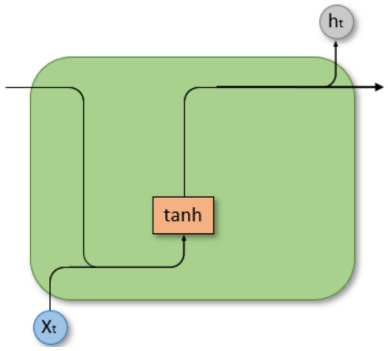


Figure 2.3: RNN unit schema: in this configuration the hidden state is updated at each step regardless of the input, leading to information bottlenecks.

Taking the inputs and the hidden state at time t produces output at time t and update the state for $t + 1$.

While RNNs are capable of retaining information from previous time steps, this ability greatly diminishes with increasing sequence length due to the way in which information can be stored in the hidden state. The model struggles to capture long-term dependencies effectively due to the vanishing gradient problem in training, gradients can shrink exponentially, making it difficult for the model to learn long-range sequences where relevant context can be far from the current input.

LSTM:

Long Short-Term Memory networks [2] (LSTMs) were introduced to address the shortcomings of RNNs, especially their inability to remember long-term dependencies. LSTMs use a more complex internal structure, highlighted in Figure 2.4, that includes gates which regulate the flow of information through the network. These gates allow LSTMs to decide what information to keep or discard at each time step, significantly improving the model's ability to capture long-range dependencies in sequential data. LSTMs have three main gates:

- Forget Gate: Decides which information from the previous time step should be discarded.
- Input Gate: Controls what new information is stored in the memory cell.
- Output Gate: Determines what information from the memory cell is output to the next time step.

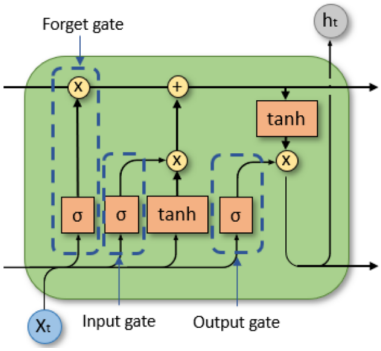


Figure 2.4: LSTM structure: gates allow to store or forget information about previous inputs and states in order to retain higher quality information about the sequence

By using gates to regulate the flow of information, LSTMs can retain useful information over longer periods, effectively mitigating the vanishing gradient problem, and are highly effective in tasks where context from earlier parts of the sequence is crucial, such as machine translation, speech recognition, and time-series forecasting. Despite their effectiveness, LSTMs are computationally more complex than simple RNNs, the additional gates and operations can make training slower and more resource-intensive.

GRU: Gated Recurrent Unit (GRU) [3] is a simplified version of LSTM that aims to achieve similar perfor-

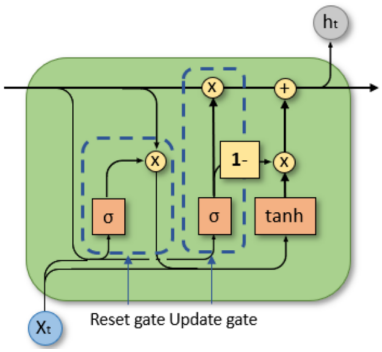


Figure 2.5: GRU structure: the simplified gates structure aims to reduce training and inference times while providing a high level information control

mance while being computationally more efficient. GRUs combine the forget and input gates of LSTMs into a single update gate (Figure 2.5). The update gate decides how much of the previous memory to retain and how much new information to add. The reset gate helps the model decide how much of the past information to forget when generating the output at each time step. As a consequence, GRUs are simpler and require fewer parameters than LSTMs, making them faster to train without sacrificing performance in many tasks. While GRUs can perform similarly to LSTMs on many tasks, they may not always capture long-term dependencies as effectively, depending on the specific data and task.

Encoder-Decoder Structure

In the context of Seq2Seq models both encoder and decoder are implemented using either RNN, LSTM or GRU. The encoder processes the input sequence step by step, updating the hidden state based on the current input and the previous state. The encoder's final hidden state serves as a summary of the input sequence, which is passed to the decoder (context vector). The decoder, also a recurrent model, then uses this context vector to generate the output sequence one step at a time, conditioned on the context vector and the previously generated elements. Seq2Seq architectures have been widely used in the industry for task such as machine translation[9], speech recognition [10] and predictive typing.

2.4 The Transformer

The Transformer architecture[4], introduced by Vaswani et al. in 2017, represents a paradigm shift in natural language processing. The key difference, compared to previous methods, is the elimination of the need for sequential processing, thus enabling parallel computation, significantly improving efficiency and scalability for large-scale NLP tasks. It has since become the foundation for many state-of-the-art models in NLP, excelling in tasks like machine translation, text generation, and question answering.

2.4.1 Architecture

Transformers mainly rely on a mechanism called self-attention, where each token in the input sequence can *attend* to other tokens, learning deep and complex contextual relationships. This allows the model to handle long-range dependencies more effectively than traditional sequence-to-sequence models based on recurrent neural networks or long short-term memory networks.

Encoder-Decoder

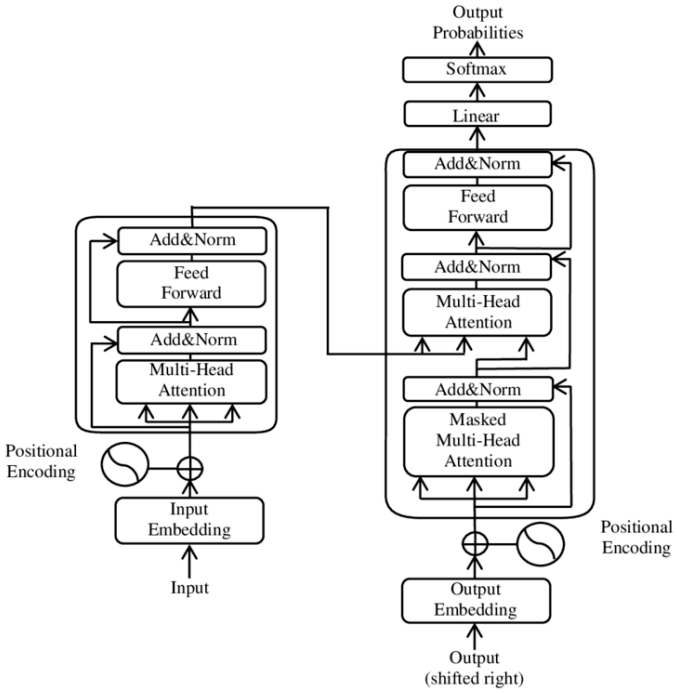


Figure 2.6: Transformer architecture with the Encoder/Decoder structure

The Transformer follows a Seq2Seq model design with two basic building blocks, an Encoder block and a Decoder block (Figure 2.6). The Encoder processes the input sequence and the Decoder generates the output

sequence.

Encoder: The encoder takes the input, which could be a sentence, and converts it into a sequence of tokens, which are words or subwords representing the basic blocks through which transformers operate on text. The sequence is then transformed into a continuous representation, called contextual embedding, that captures the semantics of the entire sequence. The encoder is composed of a stack of identical layers comprised of two sub-layers:

- Self-Attention Mechanism: This allows each word or token in the input sequence to focus on other words in the sequence when creating its representation, allowing the model to capture dependencies regardless of distance in the sequence.
- Feedforward Neural Network: After self-attention, each token's representation is passed through a position-wise feedforward network. This provides the model with additional capacity for non-linear transformation.

Each layer of the encoder also has Layer Normalization and Residual Connections to help with stability and efficient training. The output of the encoder is a sequence of vectors, each corresponding to a token in the input sequence, but each now enriched with information about the entire sequence. These vectors are then passed to the decoder.

Decoder The decoder is responsible for generating the output sequence, typically in tasks like translation or text generation. The decoder shares a similar structure to the encoder, with a few key differences:

- Self-Attention Mechanism: Just like the encoder, the decoder also uses self-attention to model dependencies within the sequence of tokens generated so far. However, the decoder is modified to prevent attending to future tokens. This is known as causal masking or masked attention, which ensures that each token can only attend to earlier tokens in the sequence.
- Cross-Attention Mechanism (or Encoder-Decoder Attention): This is where the decoder attends to the encoder's output. Each position in the decoder can attend to all positions in the encoder's output sequence, enabling the decoder to focus on relevant parts of the input sequence when generating the output.
- Feedforward Neural Network: Similar to the encoder, the decoder also passes its outputs through a feed-forward neural network.

The decoder outputs a sequence, usually in the form of logits, that represents the probability distribution over the vocabulary for each token. A final softmax function is typically applied to these logits to generate the actual token predictions.

Attention

Positional Encoding: Since the Transformer processes all tokens in parallel, it lacks an inherent sense of order in the sequence. Positional encodings are added to the input embeddings to introduce this information. These encodings are computed using sinusoidal functions:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$
$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

This design enables the model to generalize to sequences of varying lengths.

Self-Attention: Self attention is both in the encoder and decoder of the Transformer. It allows the model to dynamically focus on relevant parts of the input sequence while generating context-aware representations for each word. Unlike RNNs, which process inputs sequentially, self-attention processes the entire sequence simultaneously, capturing long-range dependencies much more effectively.

The self-attention mechanism operates through scaled dot product attention and multi head attention:

- Scaled Dot-Product Attention: This mechanism computes attention scores for every pair of words in the input sequence. The scores are derived using the dot product of query ("Q") and key ("K") vectors and scaled by the square root of the dimension of the key vector to prevent overly large gradients. The scores are normalized using the softmax function to produce attention weights.

  $$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

  Here, V represents the value vectors, which are weighted by the attention scores to produce the final output.

- Multi-Head Attention: To enhance the model's ability to capture diverse relationships in the data, the Transformer employs multiple attention heads. Each head operates independently, focusing on different aspects of the sequence and, ideally, on different semantic features. The outputs of all heads are concatenated and linearly transformed.

Masked attention follows the same rules as self-attention, the only difference is in the fact that now each token can attend only to previous tokens in the sequence.

Cross-Attention: The cross-attention mechanism is a key feature in the Transformer's decoder. This mechanism allows the decoder to attend to information from the encoder, linking the input sequence with the output sequence generation process. In the decoder layer, each position in the decoder has the ability to attend to all positions in the encoder's output. This allows the decoder to "focus" on the relevant parts of the input sequence while generating each token of the output. In cross-attention the query comes from the decoder's previous layer, while the keys and values come from the encoder's output. The ability to combine both self-attention and cross-attention in the decoder enables the Transformer to perform tasks like machine translation, where the model has to "translate" information from one language to another, maintaining dependencies between words in both the source and target languages.

Thinking from a human standpoint, when translating a sentence from one language to the other the available informations are:

- The full sentence in original language (cross attention)

– The translated sentence up to the word that has been written (masked self-attention)

2.4.2 Scalability, Applications and Limitations

Scalability

One of the most significant advantages of the Transformer architecture is that it is highly parallelizable. Since the entire input sequence is processed at once (in parallel), the Transformer benefits from modern parallel computing hardware, such as GPUs. These devices are specifically optimized for matrix operations, and the Transformer's architecture, which relies heavily on matrix multiplications in the attention mechanism and multi-layer perceptrons, makes it an ideal candidate for acceleration. Transformers also perform well with large batch sizes, as the entire sequence can be processed simultaneously without waiting for sequential dependencies to resolve. Transformers are inherently scalable because of their parallelization. With the advent of large-scale datasets and more powerful hardware, Transformers can be trained on much larger datasets than RNN-based models. Additionally, the architecture can be easily scaled up by increasing the number of layers or attention heads. This has led to transformer based models to reach unprecedented size. Starting from the 100M parameters of the original Transformer, the now relatively old GPT-3 model by OpenAI (2020) has a parameter count of 175B and the more recent Google PaLM-E (2023) [11] has a parameter count of 562B.

Applications

Another strength of the transformer is that the encoder and decoder parts of the architecture can be used independently. Based on the use case transformers are employed in encoder only mode, decoder only mode or encoder-decoder mode.

Encoder-Only Mode: In encoder-only mode, only the encoder part of the transformer is used. The encoder processes the input sequence (e.g., a sentence or text) and converts it into a sequence of embeddings that represent the contextual relationships between the words in that input, which can then be used for further tasks, such as classification and information retrieval. The most notable encoder-only model is BERT (Bidirectional Encoder Representations from Transformers). It works by pre-training on vast amounts of text and then fine-tuning on specific tasks like question answering or sentiment analysis.

Decoder-Only Mode: In decoder-only mode, transformers are used for autoregressive tasks, where the model generates output sequentially, predicting one token at a time based on previous tokens. This process is called autoregressive generation and continues until a predetermined end token is generated, completing the text. The two main applications are text generation and auto-completion or predictive typing. Examples of decoder-only transformers include most of the modern generative Large Language Models such as the GPT family from OpenAI or the Claude family from Anthropic AI.

Encoder-Decoder Mode: In the encoder-decoder mode, both encoder and decoder are used, typically for sequence-to-sequence tasks. The encoder processes the input sequence and generates a contextualized representation, which the decoder then uses to generate the output sequence. The main applications of Encoder-Decoder models are Machine Translation and Text Summarizations, since both cases are Seq2Seq problems. An example

model for this architecture is T5 [12]. It is an encoder-decoder model that treats NLP tasks as text-to-text problems. For example, in machine translation, the input is a sentence in one language (processed by the encoder), and the output is the translated sentence (generated by the decoder).

Limitations

While the transformer architecture has revolutionized natural language processing and other fields with its powerful attention mechanism and scalability, it also has several weaknesses, drawbacks, and limitations. Some of the most notable challenges associated with transformers are:

- **Computational and Memory Requirements**: Transformers require significant computational resources due to their self-attention mechanism, especially for long input sequences. The attention mechanism involves computing pairwise relationships between all tokens in the sequence, resulting in a time and memory complexity of $\mathcal{O}(n^2)$, where n is the number of tokens. This makes transformers ill-suited for resource-constrained environments and requires a maximum sequence length for models that can potentially lead to the loss of important context and information due to truncation of the input.
- **Environmental impact and accessibility issues**: Due to the high achievable parallelization but rather poor memory scaling transformer models can reach unprecedented parameter counts compared to traditional ML solutions. As a consequence training and deploying large transformer models requires a massive amount of computational power, leading to substantial energy consumption and environmental impact. The need for extreme computational resources also limits the access and development of transformers for most organizations and institutions, possibly reserving the technology to only large companies and research centers.
- **Data requirements and biases**: Transformers require large amounts of data to train effectively. Pretraining massive transformer models such as Large Language Models on vast datasets is resource-intensive and requires enormous amounts of text data. This makes transformers less accessible for tasks with limited labeled data or in low-resource languages, where collecting large corpora might not be feasible. The reliance on quantity and quality of training data also has the drawbacks of reflecting biases and problems of the data. For example, the paper 'Language Models are Few-Shot Learners' [7] states gender and racial bias from the generative language model.
- **Out-of-Distribution Data**: Transformers, like many deep learning models, may struggle with generalization to data distributions that differ significantly from the data they were trained on. This lack of robustness can lead to poor performance on tasks involving out-of-distribution data, such as when a model trained on a general corpus struggles with domain-specific language. This can also lead to models, especially generative ones, to produce plausible but factually incorrect or nonsensical answers, called 'hallucinations', because of the reliance on patterns observed in the training data without a deeper understanding of the content.
- **Lack of Explicit Memory or Long-Term Dependencies**: Despite their ability to model context within a given sequence, transformers do not have an explicit memory mechanism for handling long-term dependencies across large chunks of information, especially in tasks requiring reasoning over extended contexts. While self-attention allows the model to consider long-range dependencies within a single sequence, the model's ability to remember information across different parts of a document or across multiple conversations can be limited. In complex reasoning tasks, transformers might forget earlier details in a long text or conversation.
- **Interpretability and Explainability**: As most large scale machine learning models, transformers remain difficult to interpret. While the attention mechanism provides some insight into which words the model is

focusing on, it does not necessarily explain how the model arrived at its decisions. On top of that, the way in which transformers find and use correlations might be totally unrelated to human reasoning. Lack of transparency and explainability is a significant drawback, especially in applications like healthcare, finance, or law, where understanding why a model made a particular prediction is crucial.

2.5 Large Language Models

Large Language Models are a breakthrough technology in the field of generative AI. Based on the transformer architecture, LLMs are able to showcase human-level text understanding and text writing capabilities. This is achieved due to the impressive size of modern models and the extensive and accurate training procedure. In this section we will cover the architecture behind LLMs and how it is trained to achieve the best results. We will then focus on their applications and implementation techniques in an industrial context.

2.5.1 Decoder Only Transformer

Transformers in decoder-only mode are the core architecture for most Large Language Models. In decoder-only mode, the Transformer architecture uses only the decoder layers of the original Transformer model, which are designed for autoregressive generation of text. The input to a LLM is typically a prompt, which is a sequence of words that the model uses as context for generating text. The prompt can be a question, a simple phrase, or any incomplete sentence that the model will expand upon.

The fundamental units that a large language model uses to process and generate text are not strictly words, but tokens. These tokens can represent entire words, subwords, or even individual characters. Typically, tokens are 3 to 4 characters long, and their structure allows for the combination of tokens into more complex words and phrases. Tokenization enables the model to manage rare or out-of-vocabulary words by breaking them down into more frequent subword units. The complete set of tokens supported by a LLM is referred to as its vocabulary. The vocabulary is set at the begginning of the developement of a LLM and cannot be changed, so it requires careful consideration.

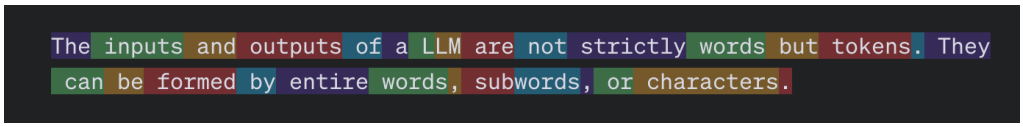


Figure 2.7: Tokenization example for OpenAI GPT-4o, highlighting how the optimal vocabulary for an LLM is composed of words and subwords of variable length.

LLM Working Example

When a sentence is input into an LLM, it is tokenized (Figure 2.7). Since the parallel computation in the transformer does not inherently account for order, a positional encoding is added to track the position of each token within the sentence. The combined token and position information is then used to encode each token in a high-dimensional embedding space that captures the sentence's full semantic meaning. The output of the decoder-only transformer is the next predicted token in the sequence, based on the input prompt. This output is represented as a vector of logits, which indicates the unnormalized probabilities for each token in the model's vocabulary. The output token is selected using various techniques, such as greedy search, where the most probable token is chosen at each step, or more advanced sampling methods, which will be discussed in a later section. The newly generated

token is then appended to the input sequence, and the process repeats. This is where the autoregressive nature of the decoder-only Transformer comes into play: at each step, the model generates the next token based on the entire sequence generated so far.

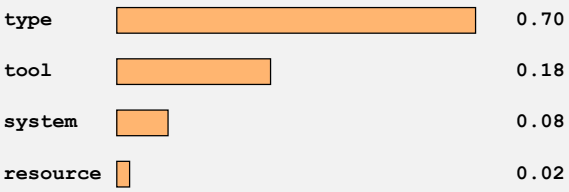
Use case example

Initial Prompt: the model starts with an initial prompt, for example

```
A large language model is a
```

This input is tokenized, and the tokens are passed through the Transformer layers.

First Token Generation: The model attends to the entire prompt, considering context, grammar, and prior knowledge and outputs a probability distribution for the next token. In this case, it might predict the word "type" with a high probability.



The generated token `type` is then added to the input sequence.

```
A large language model is a type
```

Autoregressive Generation:the model repeats the process by generating the next word, considering the entire sequence so far. The model continues generating tokens until a stop token (such as "." or a special end-of-sequence token) is reached or the maximum length is exceeded. After multiple iterations of prediction, the model outputs a complete sentence

```
A large language model is a type of artificial intelligence that
processes and generates human-like text.
```

Controlling The Generation

Despite being largely black box models, there are some key hyperparameters that can be tuned in order to control the general output behavior of an LLM. Usually, although greedy search is an option, the sampling of the next output token is performed probabilistically and, by modifying the output probability distribution it is possible to affect the resulting generated text.

Temperature: is a parameter that controls the randomness of the model's predictions. It directly affects the probability distribution from which the token is sampled during text generation. It follows an idea close to thermodynamics by modifying the probability distribution in the final layer of the LLM. Specifically, logits are divided

by the temperature value before applying the softmax function:

$$P(\text{token}) = \frac{e^{\text{logit}(\text{token})/T}}{\sum e^{\text{logit}(\text{token})/T}} \qquad \text{where } T \geq 0 \text{ is the temperature.}$$

For $T = 0$, the chosen token will always be the most probable, and so the output is completely deterministic. This is useful for tasks such as LLM evaluation, in which an accurate response is essential. For higher temperatures, the generation becomes more and more random up until the point in which the produced text loses meaning and coherence. Striking a balance between variety and stability is largely dependent on the use case scenario.

Top-k Sampling: restricts the set of candidate tokens to the top k most probable tokens based on the model's predictions. After computing the probabilities for all possible next tokens, the model only considers the top k tokens (i.e., the k tokens with the highest probabilities) and selects the next token from this reduced set.

Top-p Sampling: instead of selecting the top k tokens, the model accumulates the probabilities of tokens from highest to lowest until the cumulative probability reaches a threshold p (e.g., $p = 0.9$ or $p = 0.95$). The model then samples the next token from this nucleus of tokens whose cumulative probability is at least p.

Similarly to T, higher values of k and p make the output more variable and less predictable. These hyperparameters can be used to substantially adapt the generated text to the specific need.

2.5.2 LLM Training

Large Language Models are typically trained with a three step process:

1. Pre-training
2. Fine tuning
3. Human Alignment

Each phase has distinct objectives and is essential for enabling the model to perform a wide range of tasks.

Pre-Training

The goal of pre-training is to teach the model the basic structure of language, including grammar, syntax, facts about the world, and common knowledge. This is done using a massive corpus of text data that the model learns from in a self-supervised manner. The most common training objective is predicting the next word or token in a sequence, allowing the model to learn contextual relationships between words.

- Data Collection: pre-training relies on vast amounts of diverse text data, called Corpus. This can include books, websites, articles, code, and more. The dataset is designed to be as large and diverse as possible to capture a broad understanding of human language and to limit bias. Common sources of text include Wikipedia, news articles, literature, conversational data, scientific papers, and social media.
- Training task: Causal (Autoregressive) Language Modeling is the most common objective for LLMs. The model is trained to predict the last token in a sequence given the preceding tokens, mimicking the task of autoregressive text generation. An alternative is Masked Language Modeling and, although used more in encoder-based models like BERT, randomly masks out some tokens in the input and trains the model to predict the missing tokens. The advantage in both methods is that, from a single sentence, several training

examples can be easily obtained by either iteratively removing words at the end of the sentence or removing random words in the sentence.

- Loss Function: During pre-training, the model minimizes cross-entropy loss, which measures the difference between the predicted token probabilities and the true token values in the training data. The model's goal is to maximize the likelihood of correctly predicting the next token in a sequence.
- Training Objective: the goal of pre-training is developing self-attention, which allows the model to learn contextual relationships between tokens in a sequence. Through many layers of attention, the model refines its internal representation of the input text, learning both local dependencies (short-term relationships between adjacent words) and global dependencies (long-term relationships between distant words), thus forming a high level comprehension of natural language.

Once pre-training is complete, the model is capable of generating coherent and contextually aware text based on the patterns it learned. However, to perform specific tasks (such as question answering, sentiment analysis, or translation) proper fine-tuning is needed. Pre-training is the most time and resource consuming part of training an LLM and it is also the most expensive. It has been observed that the final output quality largely depends on the magnitude of pre-training the LLM has received.

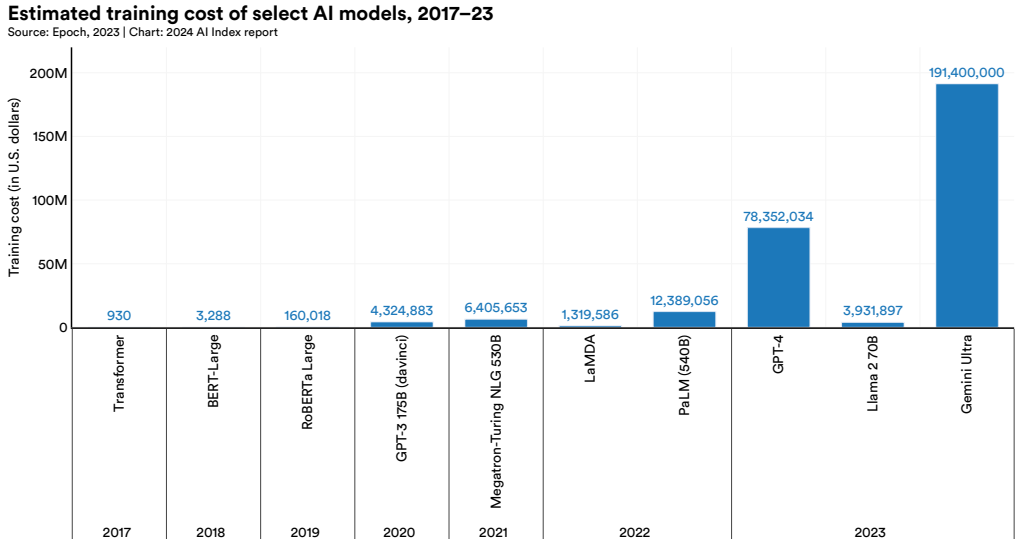


Figure 2.8: Training Cost for Transformer Based Models: training transformer based models has gotten more expensive over the years, with a bigger difference between industry and academy models - AI Index Report 2024 [13]

Figure 2.8, taken from Stanford's 2024 AI Index Report [13] not only shows how models have become progressively larger and more expensive to train, but also shows the divide between open and closed source models for the year 2023, highlighting how the increased spending capabilities of businesses translate to better performing models.

Fine-tuning

Fine-tuning involves further training on a smaller, task-specific data set, enabling the model to adapt to specific use cases. It is a form of transfer learning in which the model leverages the general linguistic knowledge gained from

pre-training and adapts it to the new task. Because of the large-scale pre-training, the model already has a strong foundation, so it can learn the specifics of the new task much more quickly and efficiently. During fine-tuning, the model takes in task-specific input and compares its predictions to the true labels in the dataset.

The LLM has now been pre-trained and then adapted to the use case through fine tuning. The last essential step is to make sure that the produced text is thorough, harmless, and, more generally, in line with human expectations.

Human Alignment

When fine-tuning an LLM, it's essential to ensure that the model responds in ways that are consistent with human goals, preferences, and safety. This is done by implementing some human feedback that can guide the LLM to produce better results. The most common techniques used are Reinforcement Learning with Human Feedback (RLHF), especially Proximal Policy Optimization (PPO) [14] and Direct Preference Optimization (DPO) [15].

Reinforcement Learning with Human Feedback (RLHF): In RLHF human evaluators are asked to manually annotate the outputs produced by a LLM through various techniques such as

- Rankings: Human annotators rank different responses that the model generates for a given prompt.
- Comparisons: The model generates multiple outputs, and human evaluators compare them to select the one that most closely aligns with their preferences.
- Direct Scores: Humans can also assign numerical scores to responses, guiding the model toward behaviors that are deemed more acceptable or preferable.

The next step is to use the feedback to construct a reward model, which quantifies how well a given response aligns with human preferences. An example reward function might be

$$\text{Reward} = \max_{\pi_\theta} \mathbb{E}_{y\sim\pi_\theta(y|x)} \Big[r_\phi(x,y) - \beta\, \mathbb{D}_{KL}\big(\pi_\theta(y|x) \,||\, \pi_{\text{ref}}(y|x)\big) \Big]$$

This formula jointly optimizes two processes

- Maximizing rewards: The part $\mathbb{E}[r_\phi(x,y)]$ is about maximizing the expected reward. The model is encouraged to choose responses that earn the highest rewards according to the reward function r, which evaluates how good or suitable the responses y are for the given inputs x.
- Minimizing deviation: The term $\max_{\pi_\theta} \mathbb{E}\Big[-\beta\, \mathbb{D}_{KL}\big(\pi_\theta(y|x) \,||\, \pi_{\text{ref}}(y|x)\big)\Big]$ is for keeping the model's behavior under control. It computes the Kullback-Leibler (KL) divergence between the model's current response strategy $\pi_\theta(y|x)$ and a reference strategy $\pi_{\text{ref}}(y|x)$. The model is penalized if its strategy strays too far from this reference, helping to ensure that while it learns to improve, it doesn't start producing unwanted or unexpected responses.

Using reinforcement learning, the model adjusts its responses to optimize for higher rewards, which are tied to human-preferred behaviors. RLHF helps produce models that can engage in more relevant and appropriate dialogues, but it comes with downsides. It's inherently unstable and costly and requires maintaining a reward model that mimics human judgment, the process is also prone to errors like reward hacking. β is a hyper-parameter of the model.

Direct Preference Optimization (DPO): DPO is a newly developed method for human alignment, which aims to directly optimize a model's responses to match human preferences, minimizing the need for complex reward models used in traditional RLHF. Instead of relying on a reward model to score the responses, DPO uses preference-based loss functions to directly train the model on which outputs are preferred over others. This can be achieved by asking the LLM to generate two responses given a single prompt and manually choosing the best response, this inherently assigns a label to the data and the problem can then be treated in a cross-entropy classification fashion. The loss function is

$$\mathcal{L}(\pi_\theta; \pi_{\text{ref}}) = \mathbb{E}_{(x,y_u,y_l)\sim D}\Big[-\log\sigma\Big(\beta\log\frac{\pi_\theta(y_u|x)}{\pi_{\text{ref}}(y_u|x)} - \beta\log\frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}\Big)\Big]$$

- y_u is the better response (upper)
- y_l is the worse response (lower)
- π_θ is the fine-tunable model
- π_{ref} is the reference model

In general $\pi_i(y_j|x)$ is the probability that model i assigns to response j given the input x, this can be done, for example, by multiplying all the logits correspondent to the tokens forming the output y_j. The model is incentivized to have the *upper* response performing better than the reference in the tuned model and the *lower* response performing worse. Also in this case β is a tunable hyper-parameter.

2.5.3 Industry Implementation

This section is dedicated to explaining the implementation of Large Language Models in everyday business use. We will explore the various ways in which these models are made available to businesses, along with the different methods by which they can be utilized and deployed by end users. Specifically, we will discuss how LLMs are integrated into business workflows, the types of platforms and tools that provide access to these models, and the potential benefits and challenges associated with their adoption in real-world applications.

Foundational Models

One of the breakthrough applications of AI are foundational models. Foundational models are extremely large, often consisting of billions to hundreds of billions of parameters and are pre-trained on vast amounts of diverse data to have a broad understanding of language, images, and other forms of data. The idea is that a single model, once trained on enough data, can serve as a foundation for many different applications, requiring little to no task-specific fine-tuning. Many of the more recent LLMs can be considered foundational models due to the large pre-training, deep language understanding and broad range of application, especially using augmentation techniques, highlighted in more details later in Section 2.6. The scale requirements of foundational models both in parameter count and data makes these models extremely resource intensive to train and maintain. For this reason the research study Stanford AI Index highlights a shift from academia to industry in the developement of large scale model, along with the increasing cost of training. Some insightful plots can be found at the end of the section (Figure 2.10 and Figure 2.11)

A direct consequence and drawback of this industry takeover is the fact that most models are now somehow restricted in access, which can either be:

- No access models: only developers can access the model (Google PaLM-E).
- Limited access models: model can be accessed only thorugh an API call (OpenAI GPT-4).
- Open access model: model is fully accessible (Meta Llama2) and editable by users.

This inevitably raises concerns regarding the accessibility of Foundational models and it is a key talking point for the future of the AI landscape.

Cloud Computing and Quantization

As was highlighted in the previous section 2.5.3 foundational models and, specifically, LLMs have computational requirements that exceed those accessible to most companies. The most common way to access these models is throught APIs and cloud computing. The models are hosted on the parent's company machines or through web services such as Amazon Bedrock [16]and the results can be retrieved by evoking a API call. Another possible solution is quantizing models, that is, converting the standard `float32 (fp32)` format of the model's weights into less accurate representations such as `float16 (fp16)` or `bfloat (bf16)` for lighter representation in memory and faster computations, provided that the processor supports computations in lower precision.

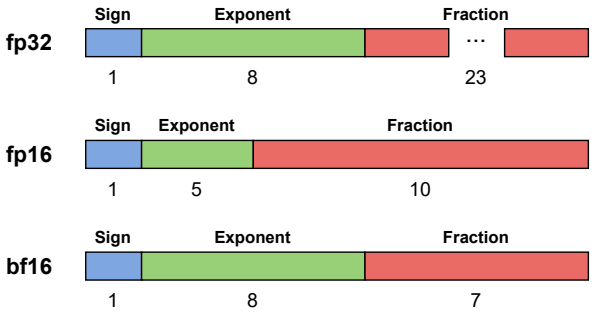


Figure 2.9: Different Floating Point Representations: bf16 has the advantage of maintaining the same exponent range of fp32, sacrificing accuracy in the fraction, while fp16 retains better accuracy at the price of a reduced magnitude range

Another technique is to convert `float32` into `int8` representation, which involves projecting the fractional values into integers using various methods, which will not be used in this research. Switching to lower precision inevitably results in diminished accuracy in computation but could still prove advantageous in resource constrained environments, where the alternative would be deploying smaller and less capable models.

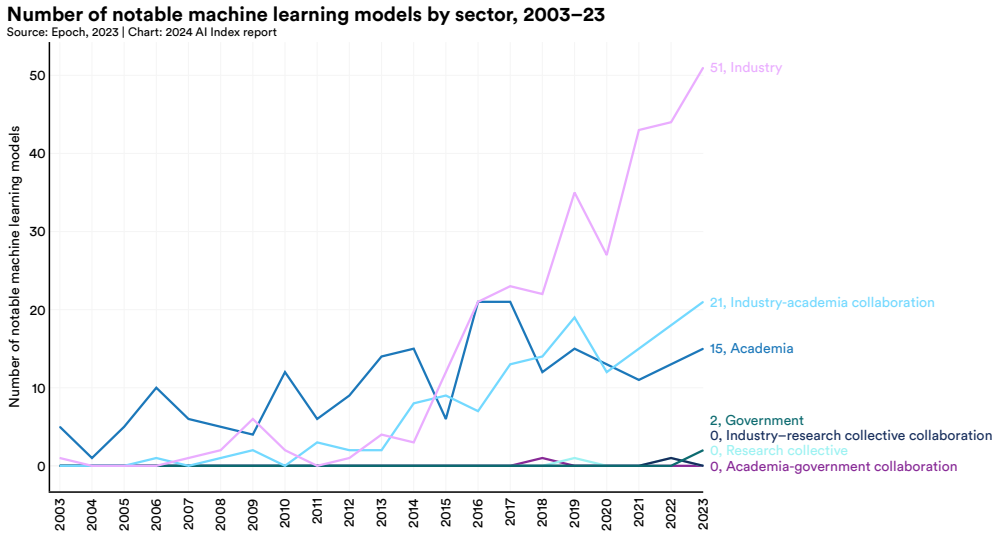


Figure 2.10: Notable machine learning models by sector: notice the increase in Industry backed models from 2017 onwards, after the release of transformers. - AI Index Report 2024 [13]

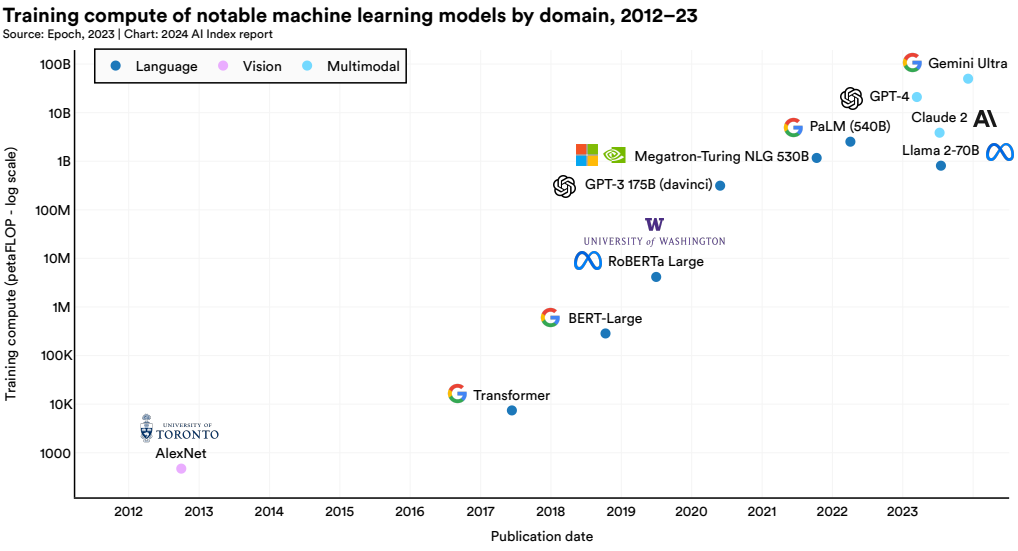


Figure 2.11: Training compute of machine learning models: training large scale models has become more and more resource intensive and expensive in the last years. Notice the log scale on the y axis - AI Index Report 2024 [13]

2.6 LLM Tuning

As has been described in the previous section Large Language Models are usually too large and computationally heavy to be fine-tuned according to the specific need. Fortunately, due to the extensive semantic understanding and knowledge acquired during training, it is possible to adapt the output of the model to our needs by crafting appropriate system prompts. This broad technique is called prompt engineering.

2.6.1 Prompt Engineering

The benefits of prompt engineering are multiple and can be resumed as:

- Improved Accuracy: By structuring prompts in a specific way, the model can focus on the relevant parts of the input and provide more accurate answers. This can involve making the prompt more explicit, adding context, or framing the question in a way that the model can better understand and respond to (query rewriting).
- Enhance Specificity: By providing clearer or more detailed instructions in the prompt, the model can be encouraged to respond in a way that aligns with the specific task at hand.
- Control Output Style: Through prompt engineering, the model can be directed to produce outputs in a particular style or tone. This can be useful when the desired output needs to be formal, casual, persuasive, concise, or detailed. It can also be used to control the length of the generated text.

A high-quality, complete prompt typically consists of two key elements: the user prompt, containing the specific instruction (e.g. "answer this question") and the system prompt, written by the developer, which defines the desired output style, adds context, and performs various functions to ensure the model generates the expected result.

Prompt Engineering - Example

In this example we are going to cover the standard system output format for the LLM family Claude developed by Anthropic, which will also be used in the research [17].
The standard parts of the prompt are:

1. Task context: Assign the LLM a role or persona and broadly define the task it is expected to perform.
2. Tone context: Set a tone for the conversation in this section.
3. Background data (documents and images): Provide all the necessary information for the LLM to complete its task.
4. Detailed task description and rules: Provide detailed rules about the LLM's interaction with its users.
5. Examples: Provide examples of the task resolution for the LLM to learn from them.
6. Conversation history: Provide any past interactions between the user and the LLM, if any.
7. Immediate task description or request: Describe the specific task to fulfill within the LLMs assigned roles and tasks.

8. Think step-by-step: If necessary, ask the LLM to take some time to think or think step by step.
9. Output formatting: Provide any details about the format of the output.
10. Prefilled response: If necessary, prefill the LLMs response to make it more succinct.

Here is the official example from the guide

```
# 1 - Task Context
Human: You are a solutions architect working at Amazon Web Services (AWS)
named John Doe. Your goal is to answer customers' questions regarding AWS
best architectural practices and principles. Customers may be confused if
you don't respond in the character of John.

# 2 - Tone Context
You should maintain a friendly customer service tone.

# 3 - Background info
Answer the customers' questions using the information provided below
<context>{{CONTEXT}}</context>

# 4 - Detailed task description and rules
Here are some important rules for the interaction:
- Always stay in character, as John, a solutions architect at AWS.
- If you are unsure how to respond, say "Sorry, I didn't understand that.
 Could you repeat the question?"
- If someone asks something irrelevant, say, "Sorry, I am John and I give AWS
 architectural advise. Do you have an AWS architecture question today I can
 help you with?"

 # 5 - Examples
Here is an example of how to respond in a standard interaction:
<example> User: Hi, what do you do?
John: Hello! My name is John, and I can answer your questions about best
architectural practices on AWS. What can I help you with today? </example>

# 6 - Conversation History
Here is the conversation history (between the user and you) prior to the
question. It could be empty if there is no history:
<history>{{HISTORY}}</history>

# 7 - Immediate task description
```

```
Here is the user's question: <question>{{QUESTION}}</question>
 How do you respond to the user's question?

# 8 - Think step by step
Think about your answer first before you respond.

# 9 - Output formatting
Put your response in <response></responses>

# 10 - Pre-filled response
Assistant: <response>
```

While it is not necessary to use all the possible properties of the system prompt is is useful to keep them in mind when crafting prompts.

2.6.2 Emerging Behaviors

An important phenomenon observed in large language models is the emergence of behaviors that were not explicitly targeted during pre-training. These behaviors, called "emerging behaviors" refer to tasks or capabilities that a model can perform effectively despite not being specifically trained on them [7]. Emerging behaviors are particularly notable because they seem to be characteristic of models that exceed a certain threshold in size and capacity. As LLMs grow larger and are trained on extensive and diverse datasets, they appear to develop novel competencies and problem-solving strategies that were not part of their initial design. This includes complex tasks such as reasoning, abstraction, multi-step problem solving, and the handling of ambiguous or incomplete information. Such behaviors challenge conventional assumptions in machine learning, where models are traditionally expected to perform tasks only if they were explicitly trained for those tasks. The most relevant emerging behaviours for the upcoming research work are zero-shot learning, in-context learning and chain-of-thought reasoning.

Zero Shot Learning

Zero-shot learning (ZSL) is the ability to perform tasks or make predictions about data that have never been seen during training. A zero-shot model is capable of generalizing to new, unseen categories or tasks without any specific examples or labels for those tasks during its training process. This contrasts with traditional supervised learning, where the model requires labeled examples for each specific task. In the context of NLP, zero-shot learning allows the model to perform tasks like classification, question answering, or translation without having seen labeled examples for the specific task during training.

Zero Shot Learning - Example

```
Message: Classify the sentiment of the sentence: 'I love my new phone!'
Answer: The sentiment of the sentence is positive.
```

The model has been pre-trained on large-scale data and likely understands the general structure of sentiment classification, even though it has not been explicitly trained on sentiment classification datasets.

Zero-shot learning is made possible by the language patterns LLMs learn during pre-training, which allows them to infer task requirements from a prompt and apply their general understanding to new situations.

In-Context Learning

In-context learning (ICL) allows a model to learn and adapt to tasks based on a given context or a set of examples presented within the input prompt. Unlike traditional training, in-context learning does not involve updating the model's parameters; instead, the model adapts to the task by conditioning on the provided examples during inference. In the case of LLMs, in-context learning leverages the model's ability to incorporate contextual information from a prompt or a series of examples within the input and perform the desired task accordingly.

In-Context Learning - Example

```
Message: Given a list of items, categorize them into fruits or vegetables.
Prompt: Fruits: Apple, Orange, Banana
        Vegetables: Carrot, Broccoli, Spinach

Answer: Cabbage is a vegetable.
```

The model looks at the few-shot examples provided and, based on the structure of the examples, learns that cabbages are typically categorized as vegetables, despite never being explicitly trained on the task.

In-context learning is extremely powerful and useful because it enables the model to "learn" from a few examples during inference, making it adaptable and flexible for various tasks without requiring re-training. Notably, ICL can also be applied to writing styles, such as rewriting sentences in a very specific way by providing examples.

Chain of Thought

Chain-of-thought reasoning (CoT) refers to the process of reasoning step by step, breaking down complex problems into simpler parts. In the specific context of LLMs, chain-of-thought reasoning involves generating intermediate reasoning steps that lead to the final answer. This approach mimics human problem-solving by encouraging the model to reason through a problem rather than just outputting an answer directly.

Chain of Thought - Example

```
Message: If a train leaves at 3 PM and travels at 60 miles per hour,
how far will it have traveled by 6 PM?
Chain of Thought:
1.The train leaves at 3 PM and we are asked how far it travels by 6 PM,
which means the time interval is 6 PM - 3 PM = 3 hours.
2.The train travels at 60 miles per hour.
3.To find the total distance traveled, we multiply the speed (60 miles per hour)
by the time traveled (3 hours).
4.So, 60 miles/hour * 3 hours = 180 miles.

Answer: The train will have traveled 180 miles by 6 PM.
```

Chain-of-thought reasoning helps the model provide a clear, transparent path to its conclusions, which is particularly useful for tasks that require logical steps or multiple inferences. This can also improve interpretability and confidence in the model's answers.

Zero Shot Learning, In Context Learning, Chain of Thought are key techniques that allow large language models to perform highly complex and adaptive tasks. They enable models to make predictions and decisions based on limited or no task-specific training, to learn dynamically from context, and to reason through problems in a structured way.

2.7 Information Retrieval

The Information Retriever, in the context of RAG, is the pipeline that allows query-document matching in order to provide additional context for the question to answer. A general information retriever is composed of three main components:

- Documents Database: contains all the relevant documents that can be useful to the LLM
- Document Embedder: converts query and documents to some vector space in order to implement a similarity search.
- Similarity Measure: built on the embedding vector space, it returns a similarity score between query,document pairs allowing to match a query with its most relevant documents.

The focus of the next section will mainly be on embedding techniques, which are the core of any retrieval system. There is a wide variety of retrieval techniques that can be employed but generally they can be divided based on the embedding space representation (sparse or dense) and the technique used in order to encode the embeddings (algorithmic or deep-learning based).

Sparse and Dense representations

The two paradigms for vector embeddings are sparse representations and dense representations.

Sparse Representations represent documents as vectors where each dimension corresponds to a specific word or term in the vocabulary. The resulting vectors are sparse because most of the elements are zero, since a document typically contains only a small subset of the vocabulary. Sparse representations are easy to interpret and computationally efficient but they can potentially become large and inefficient for large corpora. They are often used for simpler methods that fail to capture semantic relationships between words (e.g., synonyms or context), as they rely only on individual term frequencies.

Dense Representations encode words or documents as vectors in continuous, high-dimensional spaces. These vectors are dense, meaning most elements have non-zero values. Dense vectors capture deeper semantic relationships (e.g., synonyms, word analogies) by embedding words with similar meanings closer together in the vector space. Although dense representations require more computational resources to create and search through, they offer better retrieval performance in complex queries and can handle ambiguity more effectively than sparse methods.

Algorithmic-Based and Deep Learning-Based models

In Information Retrieval (IR), the distinction between algorithmic-based and deep learning-based approaches represent two different schools of thought for solving the problem of efficiently and effectively retrieving relevant information from large datasets.

Algorithmic-Based Information Retrieval involves traditional, rule-based, or statistical NLP methods which focus on manually crafted algorithms or heuristics to match and rank documents based on the query. Algorithmic IR generally relies on matching query to documents using Bag of Word (BoW) models, representing text as a collection of terms without considering the order of words; the more terms in a document that match the query, the higher its relevance score. Common models include TF-IDF (Term Frequency-Inverse Document Frequency) and BM25, which are widely used for document ranking and will be covered in a later section. These models are generally easier to understand, implement, and interpret. They work well for straightforward tasks where document relevance is mostly determined by keyword matching and tend to be computationally less expensive compared to deep learning models, making them suitable for real-time applications and large-scale indexing. On the other hand, algorithmic models typically lack a deep understanding of the meaning or context behind the words. They are based purely on surface-level features like term frequency and do not capture word semantics, synonyms, or context. The lack of context in models like TF-IDF and BM25 means that they may not capture long-range dependencies in text or complex queries.

Deep Learning-Based Information Retrieval: Deep learning-based approaches to Informaion Retrieval leverage neural networks, particularly transformer models, to automatically learn representations of text that can be used to retrieve relevant documents. These methods have been particularly effective in improving the accuracy of search systems by modeling the semantic meaning of queries and documents, going beyond simple term matching. Modern deep learning models, such as transformers, can even account for the context of words in a query or document and are able to efficiently capture even long-range dependencies, allowing them to understand complex queries and documents that involve multiple layers of meaning, without the need for manually crafting features. The resulting retrievers boast major performance improvements over traditional methods, however, they also present significant challenges. Deep learning models require significant computational resources, including large datasets and powerful hardware both for training and inference. This can make them expensive to deploy and maintain and inevitably slower in retrieval.

In Information Retrieval, algorithmic-based approaches like TF-IDF and BM25 are still widely used for their efficiency and simplicity, especially in situations where computational resources are limited or where search is based on keyword matching. However, deep learning-based approaches, particularly transformer models like BERT and DPR [18], have significantly advanced the field by providing semantic understanding and enabling more sophisticated search capabilities, particularly in handling complex queries and understanding the context. The next section will be dedicated to the comparison between dense and sparse retrievers that are relevant for this study.

2.7.1 Dense Retrieval

In dense retrieval the embedding shape is a fixed length dense vector, meaning that elements are tipically non-zero and the information is spread across all dimensions of the embedding space. At the core of modern, high performance, dense retrievers often lie transformer based architectures in encoder-only mode.

BERT

The first breakthrough architecture for transformer embedders was BERT (Bidirectional Encoding Representation from Transformers), developed by Google in 2018 [6]. It is an encoder-only transformer that produces token embeddings given an input sequence.

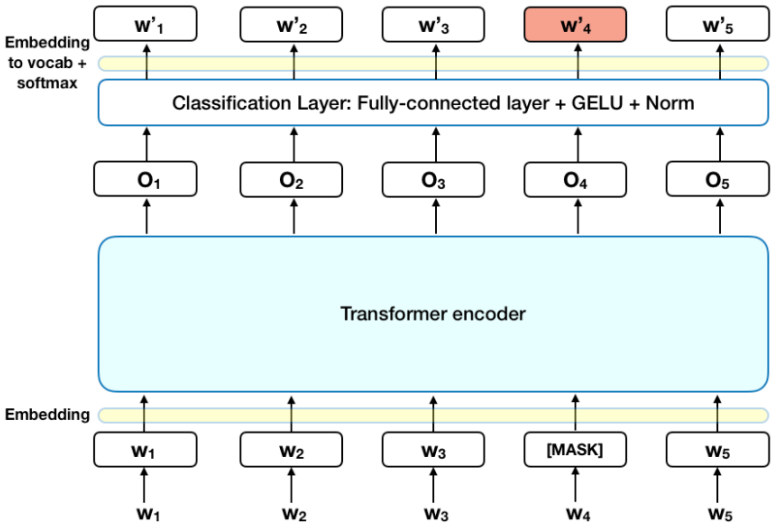


Figure 2.12: BERT: The input is tokenized w_i and embedded through a transformer encoder, producing word embeddings o_i. Optionally, embeddings can be reverted back to words w_i^1

BERTs architecture is based on four blocks shown in Figure 2.12:

- Tokenizer: Converts the input text into tokens
- Embedder: Converts tokens into numerical embeddings
- Encoder: Transformer encoder with bi-directional self attention. Unlike for LLMs here it is possible and encouraged to 'look' at the entire sentence before producing attention scores
- Task Head (optional): Returns the final representation vectors back to one-hot encoded tokens by producing a predicted probability distribution over the token types. This step is necessary during training, but is usually omitted for downstream tasks.

BERT was pre-trained on two different self-supervised tasks

- Masked Language Modeling: Given a complete sentence, it was passed to the model either with a missing word, a replaced word or just as it was. The task of the model was to understand if any word needed replacement, and select the correct replacement accordingly
- Next Sentence Prediction: Given two sentences, the task was to predict wether they would come sequentially in a piece of text, that is, if they were contextually and semantically relevant to one another.

After pre-training, much like what happens for LLMs, it is possible to fine-tune the model for the specific task. In the original research paper various fine-tunings were performed such as Sentiment Analysis, Sentence classification, Question Answering and Speech tagging. The strength in BERT, as for most transformers models, lies in the increased semantic understanding which allows for higher quality embeddings.

Sentence-BERT

Sentence-BERT (SBERT) [19] is an adaptation of the BERT model that has been specifically designed to generate meaningful sentence embeddings. While BERT focuses on understanding individual words, SBERT takes a different approach by creating fixed-size sentence embeddings that capture the overall meaning of a sentence. The core idea behind SBERT is to fine-tune BERT, or similar transformer models, on tasks that involve comparing sentence pairs, such that semantically similar sentences are placed close together in a vector space. SBERT typically employs a Siamese network architecture, where two identical BERT models with shared weights process sentence pairs independently and then compute a similarity score based on the distance between their embeddings (Figure 2.13. This differs from other deep-learning based architectures, such as DPR (Dense Passage Retrieval) [18] in which query embedding and document embedding are performed by two independent BERT or BERT-like encoders.

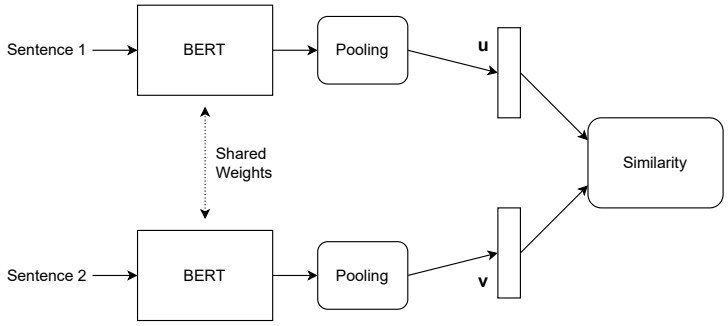


Figure 2.13: Sentence-BERT: pairs of sentences are encoded with an identical BERT model and the resulting output is pooled to a fixed embedding size. The similarity score is computed between the vectors of the sentence embeddings

The central operation in shifting from word embeddings to sentence embeddings is the pooling operation after the BERT encoding which allows for a dimensionality reduction:

$$\text{pool}\big(\text{dim} = (\text{num. tokens}, 768)\big) \rightarrow \text{dim} = 768$$

768 is the embedding dimension of the original BERT but can vary depending on the chosen transformer model.

Pooling techniques can be multiple:

- Max Pooling: the maximum value of each token logit is kept
- Mean Pooling: the mean value of each token logit is extracted
- CLS pooling: value of the CLS token at the beginning of each sentence as encoded by BERT

Based on the experiments performed by the paper's authors, the Mean Pooling is the best performing technique and thus it is used. Training the model means adjusting the weights in the Siamese network such that the similarity score between related sentence is higher than similarities between unrelated ones. Once the model has been trained, only one of the encoders is needed to compute sentence embeddings. These embeddings can then be stored and used in real-time applications, such as search engines, recommendation systems, or clustering tasks.

2.7.2 Sparse Retrieval

The traditional approach to performing query-document search is through lexical models and Bag-of-Words methods, which focus on the use of words (or terms) as the fundamental units for indexing and retrieval. At the heart of BoW methods lies the simplifying assumption that a document's content can be adequately represented as an unordered collection of words, disregarding grammar and word order but preserving the frequency of occurrence of each term. This simple representation forms the basis for several text-retrieval models, such as Term Frequency, Term Frequency-Inverse Document Frequency, and BM-25.

Term Frequency Methods

Term Frequency: The TF model counts how frequently a term appears in a document. The basic idea is that the more often a term appears, the more relevant the document might be for that term. However, raw term frequency can be misleading because terms appearing frequently across all documents may not be as important for distinguishing between documents.

Use case example

Suppose we have a document database composed of just two sentences:

Entry1: "This article is about information retrieval"
Entry2: "Information retrieval is the process of obtaining information resources that are relevant to an information need"

The first step is to construct the words vector space:

Key	Value	Key	Value
0	about	9	information
1	are	10	relevant
2	an	11	retrieval
3	article	12	resources
4	is	13	the
5	obtaining	14	this
6	of	15	that
7	need	16	to
8	process		

Each unique term is an independent dimension of the space, in this case the resulting space is 17 dimensional. Applying TF mapping:

$$\begin{aligned} TF(\text{Entry1}) \rightarrow &\quad [1,0,0,1,1,0,0,0,0,1,0,1,0,0,1,0,0] \\ TF(\text{Entry2}) \rightarrow &\quad [0,1,1,0,1,1,1,1,1,3,1,1,1,1,0,1,1] \end{aligned}$$

Term Frequency-Inverse Document Frequency (TF-IDF): The TF-IDF model enhances the basic TF model by incorporating the Inverse Document Frequency (IDF), which adjusts for the fact that some terms are very common and appear in many documents. The TF-IDF score is calculated by multiplying the term frequency with the inverse of the document frequency, defined as:

$$\text{Term Frequency} \quad \text{TF}_{\text{word,document}} = \frac{\text{occurencies of the word in the document}}{\text{total words in the document}}$$

$$\text{Inverse Document Frequency} \quad \text{IDF}_{\text{word}} = \log\left(\frac{\text{total number of documents}}{\text{number of documents containing the word}}\right)$$

IDF is a penalty term for recurring terms that are likely to be of little significance in the text.

Use case example

In the same case as before, the IDF mapping can be computed (shown rounded to integers):

$$\text{IDF} \sim [1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1]$$

The most important takeaway is that, for the terms appearing in both documents ('is', 'information', 'retrieval') the significance drops to 0. Applying now also TF:

$$\begin{aligned} \text{enc}(\text{Entry 1}) &\rightarrow \quad [1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0] \\ \text{enc}(\text{Entry 2}) &\rightarrow \quad [0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1] \end{aligned}$$

BM-25: Best Matching 25 (BM-25) is a more advanced probabilistic model that refines the ideas of TF and IDF. The formula for BM-25 is:

$$BM25(D, Q)_{b,k} = \sum_{i=1}^{n} IDF(q_i) \frac{(k+1) \cdot TF(q_i, D)}{TF(q_i, D) + k\big(1 - b(1 - \frac{|D|}{\langle |D| \rangle})\big)}$$

where: Q_i for $i = 1, ..., n$ are the words contained in a query
$|D|$ is the document length and $\langle |D| \rangle$ the average over all documents
b, k are tunable hyper-parameters

In addition to TF and IDF, BM-25 considers the length of documents and includes a saturation effect to adjust the contribution of term frequency for very high frequencies. The model also introduces parameters for tuning, such as the b parameter, which controls the effect of document length and k, which controls the term frequency scaling. Despite its simplicity, BM-25 is widely used in the field of information retrieval mainly for the reasons that it is a very numerically efficient algorithm, does not require prior traning, and can be hyper-parameters tuned to the specific use case scenario. Still, the underlying assumption behind Bag-of-Words algorithms results in a performance ceiling that has been surpassed by more advanced models.

SPLADE

SPLADE (Sparse Lexical and Expansion Model for Information Retrieval)[20] is a model designed to improve the effectiveness and efficiency of information retrieval system. It introduces a novel approach to combining sparse and dense representations in information retrieval tasks, with a focus on achieving efficient sparse retrieval representations while leveraging the power of dense neural models. The model achieves this by ensuring that the learned embeddings emphasize important lexical features while also capturing broader contextual meaning. The key idea behind SPLADE is lexical expansion, a technique to introduce more relevant terms (related or contextually similar terms) into the sparse representation. SPLADE starts, like most dense embedders, with a BERT encoding, which is then transformed into a sparse representation with the added expressive power of the underlying NN architecture. SPLADE includes a lexical expansion step, which ensures that the sparse representation is enriched with additional relevant terms. This is done in a way that ensures the final representation captures both the precise wording and the broader semantic meaning of the input. The resulting embeddings can then be used

for query document matching.

The advantage for SPLADE lies in retrieval efficiency. Dense retrieval models like BERT can be computationally expensive, as they require the comparison and storage of dense embeddings. SPLADE, by producing sparse representations, makes retrieval more efficient, reducing the computational overhead associated with comparing dense vectors. This can be particularly important for large-scale retrieval tasks, where efficiency is crucial.

3

Domain Adaptation of RAG Systems

3.1 Problem Definition

When dealing with private or domain-specific information, it is not sufficient to rely only on a general purpose LLM, regardless of how powerful it might be, to generate meaningful and true answers. To enhance the performance in answering it is essential to develop a retrieval pipeline that can provide the necessary context to produce an answer, thus implementing a RAG system. Much like what has happened for LLMs, there is a wide variety of cloud-based high performance embedding and retrieval models that can be used in order to implement a RAG system. The advantages of cloud based APIs are:

- Ease of deployability
- High, state of the art performance

On the other hand they have some limitations, such as:

- Increased latency in embedding
- Cost per call
- Models not fine-tuned to specific needs
- Large embedding dimension

Most companies offer tailor made fine tuning of models in order to dramatically increase the performance, however, this also has notable drawbacks. Fine tuning on large-scale third party models can be overly expensive and privacy/compliance complications might arise when sharing classified documents with third party entities. The proposed idea is to consider smaller models, that can run locally, to be fine-tuned on specific company documents. This would solve most of the challenges with cloud-based embedders:

- Low latency
- Lower cost
- Increased performance due to fine-tuning
- Smaller, lighter embeddings
- Business or private data does not need to be shared with external entities.

This approach, however, calls for the creation of a dedicated annotated query/document dataset on which the model can be fine-tuned. The traditional way to create such a dataset is human annotation, which comes with its own set of drawbacks:

- Human annotation is slow and expensive
- Annotators need to be carefully trained in query creation and validation
- Human annotation can be subject to heavy personal bias
- Privacy/compliance concerns may arise from the annotators accessing business documents

In recent years LLMs have shown a human level understanding of sentence composition and perform remarkably well in evaluation. The proposed framework will employ high performance LLMs for generating, validating and evaluating a synthetic QA dataset. The resulting artificial dataset can then be used to locally fine-tune models that rival and even surpass current state-of-the-art (SOTA) retrieval models in End2End evaluation on the specific domain. The latter sections will describe in detail each step of the pipeline

3.2 Artificial Questions Generation

The core focus of this thesis involves the creation of a synthetic Question-Answer (QA) pairs dataset starting from private company documents, in this section each step of the generation and validation pipeline is accurately described and presented and the most relevant results are shown.

Documents Analysis

The provided documents used for the research are internal documents of the company Omnys S.r.l. regarding a variety of specific topics mainly regarding safety policies, risk assesments and internal company management. For security reasons, they cannot be released, but very limited sections will be presented during the research.

3.2.1 Text Extraction and Segmentation

The process of creating a synthetic dataset from `.pdf` files involves several key steps. First, the `.pdf` files are converted to text sections using models from the Unstructured library and the output is saved as a markdown file `.md`, chosen for its readability and ease of manipulation. After some basic cleanup, the markdown file is used to subdivide the full document corpus into smaller, more manageable sections. These sections, referred to as 'passages' or 'sections', are created by splitting the markdown along its natural divisions, encoded by the characters `#`. This

subdivision is particularly important when working with LLMs, as it allows for more effective processing of the text data. By breaking the document into smaller chunks, LLMs can better handle the information, reducing the risk of errors or inconsistencies in their output while maintaining a coherent piece of text.

A database can be created with the passages in the schema:

- document name: name of the parent document
- section name: header of the section
- section content: content of the section

Document section example

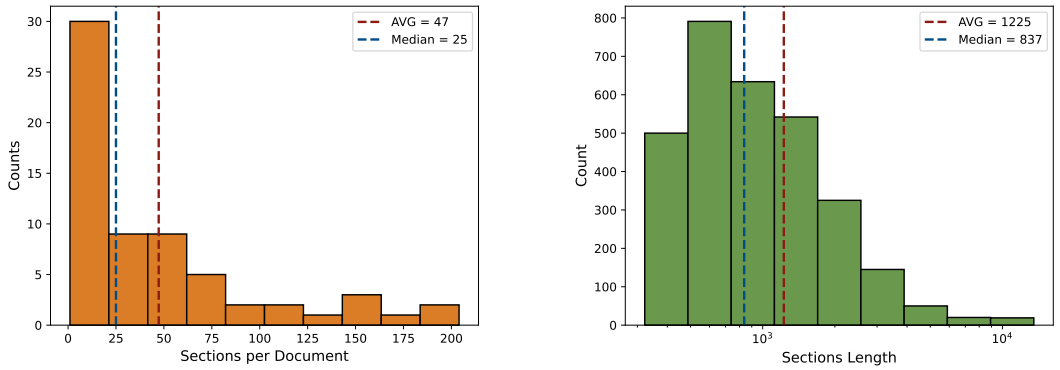
```
Document: ISO_IEC_27000_2018.md
Header: Management System

Content: A management system uses a framework of resources to achieve an
organizations objectives The management system includes organizational
structure policies planning activities responsibilities practices procedures
processes and resources In terms of information security a management system
allows an organization to
a satisfy the information security requirements of  customers and other
stakeholders
b improve an organizations plans and activities
c meet the organizations information security objectives
d comply with regulations legislation and industry mandates and
e manage information assets in an organized way that facilitates continual
improvement and adjustment to current organizational goals
```

The data set consisted of 64 `.pdf` documents that were extracted and divided into 612 unique passages. In Figure 3.1 some highlights of the extracted sections are presented. Most documents are short, containing under 25 subsections, with a tail of long documents resulting in an average of 47 passages per document (Figure 3.1a). The average passage is around 10^3 characters long, which translates to roughly 250 tokens, well in the ideal operating range for LLMs (Figure 3.1b).

3.2.2 Generation and Validation

Once the database has been created, it is possible to use it to generate synthetic but human-like questions about the content of documents. To simulate a realistic chatbot scenario, the questions are designed to be cross-lingual, meaning the language of the question differs from that of the documents. For instance, a question might be in German, the retrieved document in Italian, but the answer must be generated in German. This approach presents challenges for both the retriever and generator models, as they must effectively handle the language discrepancies while maintaining accuracy and relevance in their responses. This cross-lingual aspect adds complexity to the task but also enhances the robustness and versatility of the resulting system.



(a) Extracted sections per documents

(b) Average section length

Figure 3.1: Features of the extracted dataset: documents have varying length with most of them being under 25 passages long with longer documents containing up to 200 sections. Section length is also variable but it is concentrated around 1000 characters, or 250 tokens approximately per passage.

Given the complexity of the task, particularly its cross-lingual requirements, a high-performance model is essential at this stage. The LLM chosen for this purpose is `Anthropic Claude v3.5`[21], which is part of a series of closed-source LLM models developed by Anthropic AI and deployed in Amazon Bedrock as a Cloud Service. This model was selected due to its advanced capabilities in handling complex, multilingual tasks. However, the selection of an appropriate model is only part of the solution. Equally important is the implementation of a carefully curated generation and validation pipeline (Figure3.3). This pipeline plays a crucial role in maximizing the quality of the generated dataset.

Questions are generated in eight languages with different resource levels:

1. High resource languages: English, Chinese
2. Medium resource languages: Italian, French, German, Spanish
3. Low resource languages; Japanese, Vietnamese.

The resource level of a language is determined by the data availability, quality, and general usage in ML systems.

Generation

The first step in the pipeline is question generation. Given a passage as context the LLM is tasked with

1. Understanding the meaning of the piece of text
2. Selecting the most informative parts of the text
3. Formulating a meaningful question in one of the selected language

To properly understand the actions performed by the LLM it is useful to analyze the prompt step by step, recalling from the prompt engineering section the prompt is divided in system prompt and user prompt.

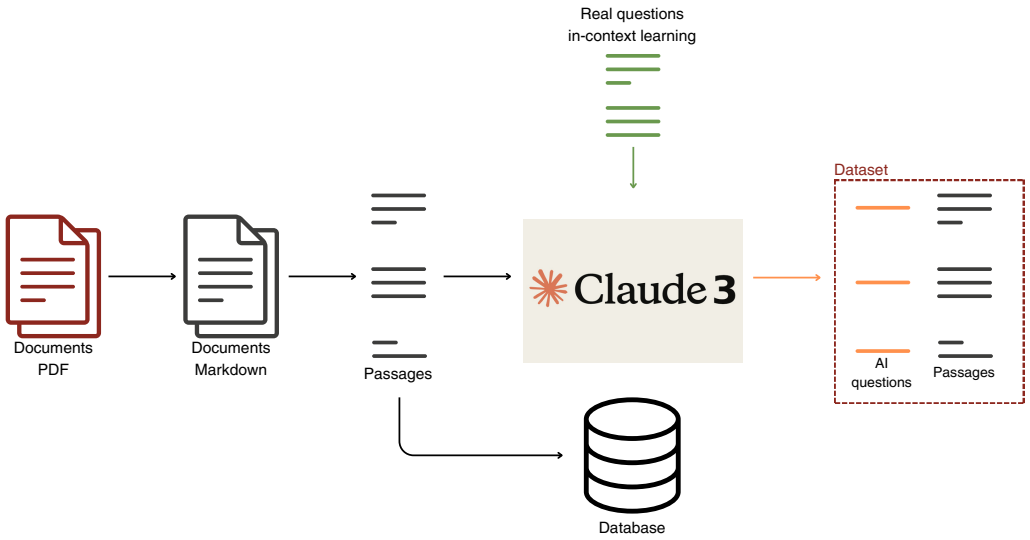


Figure 3.2: Text extraction and question generation schema: pdf documents are converted to markdown and split into passages. These passages become entries of a database and are used to generate synthetic QA pairs using a LLM.

Questions Generation - PseudoCode

```
INPUTS:
    - Language
    - Information about the document (Name, Section, Content)

OUTPUTS:
    - Generated Question

REQUIREMENTS:
    - Questions should be conversational and informal in tone,
    strongly avoiding lengthy or overly specific phrasing.
    - They can be formulated either as questions or as assertive instructions
    - Avoid more than one sentence per question,
    try to make a question shorter that 10 words.
    - Avoid referencing specific parts of the document in the question
    - Questions should be focused on open-ended information
    (not yes/no or confirmatory questions).
REMEMBER: generate questions based only on the information contained
in the excerpt, do not rely on any previous knowledge.
REMEMBER: if you are not able to generate any meaningful question, return 'NA'.
```

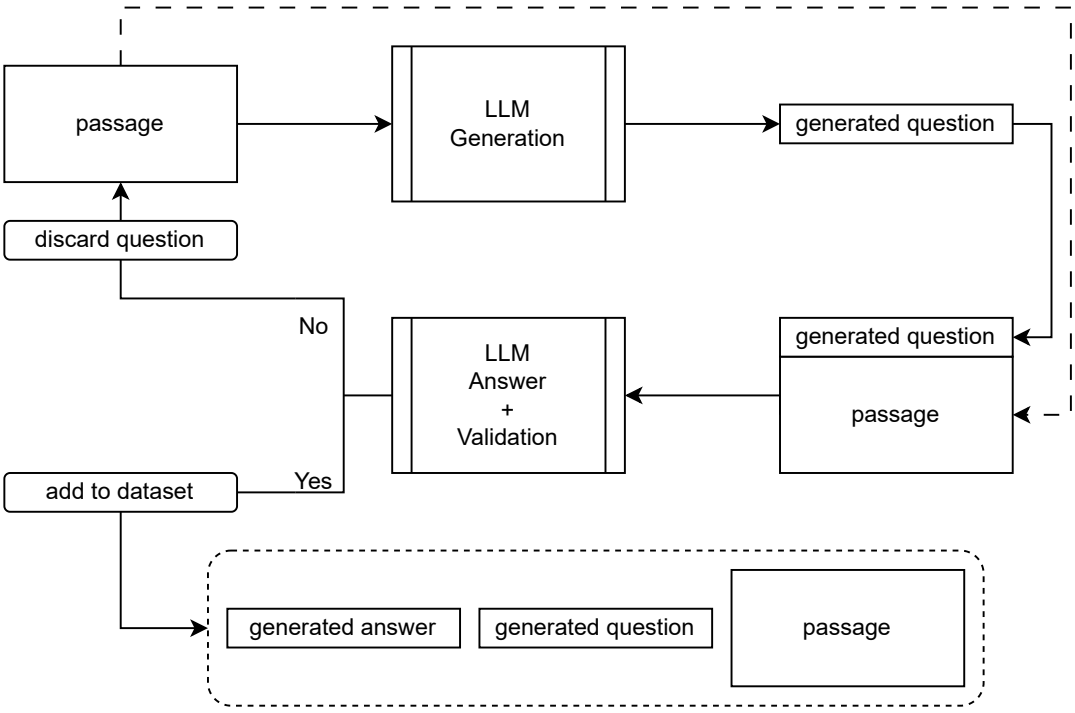


Figure 3.3: Generation+Validation Pipeline: Questions are generated based on the content of passages. In a second, independent, LLM call the question is validated with the passage as context and, depending on wether an answer is generated, is kept or discarded.

Validation

To ensure the quality and relevance of the generated questions, a robust validation procedure is implemented. This process involves two separate interactions with the Large Language Model. Initially, the LLM generates a question based on the given document. Subsequently, in an independent call, the LLM is tasked with generating the answer to this question, using the original document as context. The independence of these two calls is crucial, as LLMs lack memory between interactions. This approach allows the second call to serve as an unbiased evaluator of the quality and answerability of the generated question. By leveraging the LLM's capabilities in both question generation and answer retrieval, this validation procedure effectively assesses the meaningfulness and coherence of the synthetic questions.

The full validation prompt can be found in the appendix but the core features are:

Questions Validation - PseudoCode

```
INPUTS:
    - Generated Question
    - Language
    - Information about the document (Name, Section, Content)

OUTPUTS:
    - Generated Answer

REQUIREMENTS:
    - Answers should be clear and accurate
    - Keep answers as short as possible

    IMPORTANT: the answer must be provided in the same language as the question
    IMPORTANT: the questions were created based on the content of the documents
    so you should find the answer
    IMPORTANT: if you think the question is malicious or could compromise
    the safety of the company give as an answer 'denied_safety'
    IMPORTANT: if you can't find the answer in the documents
    give as an answer 'NA'
```

The validation process continues with a critical assessment of the generated question-answer pairs. If the model successfully produces a meaningful answer to the generated question, both the question and its corresponding answer are incorporated into the dataset and the generated answer will serve as the 'true' answer for later evaluation. However, if the answer is deemed inadequate or irrelevant, the pair is rejected, and a new question is generated. This process is repeated until one of two conditions is met: either the target number of questions per document is achieved, or a predetermined maximum number of errors is reached. In the latter scenario, the generation process for that particular document is aborted.

For each passage 5 questions were generated and validated in randomly selected languages. As a result, from the 612 extracted passages 3060 artificial questions were generated. Out of those, 33 were discarded due to poor formatting (1%) so the final dataset size comprised of 3027 questions. The language split is highlighted in Table 3.1

The language imbalance could be due to the higher likelihood of successful generation for high resource languages. Finally, a example of question generation and validation is shown:

Generated QA pairs	
Language	Questions
German (de)	312
English (en)	439
Spanish (es)	402
French (fr)	345
Italian (it)	361
Japanese (ja)	370
Vietnamese (vi)	337
Chinese (zh)	461

Table 3.1: Language Split

Example Generation + Validation

```
#CONTEXT
Document: ISO_IEC_27000_2018.md
Header: Management System

Content: A management system uses a framework of resources to achieve an
organizations objectives The management system includes organizational
structure policies planning activities responsibilities practices procedures
processes and resources In terms of information security a management system
allows an organization to
a satisfy the information security requirements of  customers and other
stakeholders
b improve an organizations plans and activities
c meet the organizations information security objectives
d comply with regulations legislation and industry mandates and
e manage information assets in an organized way that facilitates continual
improvement and adjustment to current organizational goals

#GENERATED QA
Question: What are the benefits of an information security management system?
Language: en

Answer: The main benefits of an information security management system are:
1) Satisfying information security requirements of customers and stakeholders,
2) Improving organizational plans and activities,
3) Meeting information security objectives,
4) Complying with regulations, legislation, and industry mandates,
5) Managing information assets in an organized way that facilitates continual
improvement and alignment with organizational goals.
```

Train-Test Split

The train-test split is done on a document-by-document basis to prevent cross-contamination between datasets. This means that questions and passages from a single document can only appear in one of the three datasets. By doing so, we avoid the risk of having similar questions or questions about the same document appear in both the training set and the evaluation sets. If this were to happen, the model might just memorize the QA pairs it already knows, rather than learning a more general representation. The percentage split is performed based on the number of documents with 80% of the documents reserved for training, 10% for validation and 10% for the test set with a seeded random sample.

	Documents	Passages	Questions
Train	51	442	2190
Validation	6	75	368
Test	7	96	469

Table 3.2: Train/Test Split: Documents can be part of only one dataset to avoid cross-contamination.

3.3 Embedders Fine Tuning

After the dataset has been created the retrieval pipeline is set up. The most important part is the document embedder, for this reason a variety of methods has been tested. Sparse retrievers are shown in Table 3.3. The three BM-25 variations are refinements [22] on the original BM-25 model described previously, in particular:

- BM-25L Adds a lower-bound correction term L to the length normalization factor. It prevents short documents from getting high scores due to relative high term frequency, improving ranking consistency.
- BM-25Plus Adds a positive constant δ to term frequency. Prevents penalizing rare terms with low frequencies, improving ranking when such terms are relevant.
- BM-250kapi Dynamically adjusts k_1 parameter for task-specific optimization. Provides more flexibility and performance tuning for different retrieval tasks or document collections.

Dense retrievers are presented in Table 3.4.

- `paraphrase-multilingual-MiniLM-L12-v2` (MiniLM L12): is a multilingual model developed by the `sentence-transformer` team [19] based on the MiniLM model by Microsoft [23]. Its most distinct feature is the reduced embedding dimension 384, that results in faster inferences and lighter representations.
- `paraphrase-multilingual-mpnet-base-v2` (MPNET): is another multilingual model trained by `sentence-transformer` [19] which uses the more advanced XLM- Roberta model[24]. It features an embedding dimension of 768.
- `bge-m3` (BGE): is a multilingual model developed by BAAI-Beijing Academy of Artificial Intelligence [25]. It features a large 1024 dimension embedding.

SPARSE RETRIEVERS	
Model Name	Type
BM-25L	Lexical
BM250kapi	Lexical
BM-25Plus	Lexical
SPLADE	Deep

Table 3.3: Retrieval Models - Sparse

— `cohere.embed-multilingual-v3` (Cohere [26]): Is a closed source model developed by Cohere Inc. for multilingual text embedding. It boasts an embedding dimension of 1024 and it is widely considered the state of the art in retrieval tasks.

DENSE RETRIEVERS				
Model Name	Emb. Dimension	Context Window	Parameters	Fine-Tunable
MiniLM L12	384	128 Tokens	33M	YES
MPNET	768	128 Tokens	270M	YES
BGE-M3	1024	768 Tokens	580M	YES
Cohere	1024	2048 Characters	Unknown	NO

Table 3.4: Retrieval Models - Dense

Base Models

The first stage is to evaluate all the models in their default configuration in order to set a performance baseline. The ranking ability is tested on questions from the test set while the available document corpus is made of all the available documents in the dataset, this is done to avoid unwanted effects from the small document sample size of the test set by itself. Being a ranking task the evaluated metrics are ACC@1 to test the ability of the model to extract the true document as the first, while MAP@10 and NDCG@10 are more faithful and accurate representation of ranking performance (more details in section 2.2.1).

Some insights from the first evaluations are:

1. The difference between dense and sparse methods is very noticeable. This is possibly highlighted by the cross-lingual nature of the dataset, for which keyword matching, even with lexical expansion, becomes increasingly difficult.
2. For sparse encoding the simpler BM-25L is by far the worst performer, with the two other variants BM-25Plus and BM-250kapi performing much better and very similarly to each other.
3. SPLADE is, expectedly, the best performing sparse model due to its added expressive power given by lexical expansion.

Retriever Baselines - Test Set			
Model Name	ACC@1	MAP@10	NDCG@10
BM-25L	0.034	0.068	0.092
BM-25Plus	0.134	0.169	0.191
BM-250kapi	0.130	0.165	0.187
SPLADE	0.181	0.244	0.279
MiniLM L12	0.424	0.526	0.576
MPNET	0.456	0.550	0.602
BGE-M3	0.507	0.616	0.672
Cohere	**0.552**	**0.658**	**0.710**

Table 3.5: Base Retrievers Results: The performance difference between sparse and dense methods confirms the capability of transformer based architectures for sentence embeddings. Larger models perform better compared to lighter ones.

4. For dense encoders it generally holds that larger models lead to better results, with a constant increase in accuracy and precision as the embedding dimension and context length increase.
5. MiniLM L12, despite its limited embedding dimension performs in a very satisfactory way.
6. Cohere is the best performer, as industry standard it is not unexpected.

Multiple Negatives Ranking Loss Training

After the baselines have been set it is time to leverage the synthetic dataset in order to fine-tune the sentence embedders. What the training procedure amounts to is to adapt the representation space in order to accurately reflect the semantic similarities of queries and documents for the specific data set. The appropriate approach for this problem involves training with query-document pairs and MultipleNegativesRankingLoss (MNRL) [27].

MNRL is a specialized loss function used for training sentence transformers models for tasks such as information retrieval, ranking, and semantic similarity. It is particularly effective when dealing with large-scale datasets where the objective is to differentiate between relevant and irrelevant pairs of sentences. This has a direct parallel with retrieval, in which the algorithm will need to discriminate between relevant and irrelevant documents for a specific query in order to return the most relevant results.
The synthetic dataset was created with a `(query,document)` pair structure, with the query being the generated question and the document the passage used for generating said question. The idea is that the similarity between query and document of origin should be larger than the similarity between the query and any other document.

In the training procedure, given a batch of sentences, each query is paired with its most relevant document to form a positive `(query,document)` pair (label $l = 1$) while every other document in the batch is used to create negative `(query,document)` pairs (label $l = 0$). This is an example of self supervised learning, in which labels are inferred from the data itself rather than manually annotated (Figure 3.4).

$$
\begin{array}{rcll}
 & (q_2, D_2) & & (q_1, D_1) \quad l = 1 \\
 & (q_3, D_3) & & (q_1, D_2) \quad l = 0 \\
(q_1, D_1) \leftrightarrow & (q_3, D_3) & \rightarrow & (q_1, D_3) \quad l = 0 \\
 & \dots & & \dots \quad \dots \\
 & (q_n, D_n) & & (q_1, D_n) \quad l = 0
\end{array}
$$

Figure 3.4: Creating self supervised labels in MNRL loss. Starting from a batch of (query,document) each query is paired with the corresponding correct document with label 1 and with each other document with label 0

MN-Rank Loss has shape:

$$\text{MNRL}(x, y_1, ..., y_n) = -\alpha \big[\text{sim}(x, y_1) - \log \sum_{j=2}^{N} e^{\text{sim}(x, y_j)}\big] = -\alpha \log \frac{e^{\text{sim}(x, y_1)}}{\sum_j e^{\text{sim}(x, y_j)}}$$

With α being a scaling factor used with normalized embedding (by default $\alpha = 20$). In our specific case a batch of size n is formed by `query` q, `positive` document p and $m = n - 1$ `negative` documents $(n_1, ..., n_m)$

$$\text{MNRL}(q, p, n_1, ..., n_m) = -\alpha \log \frac{e^{\text{sim}(q,p)}}{e^{\text{sim}(q,p)} + \sum_{i=1}^{m} e^{\text{sim}(q, n_i)}}$$

Where:

- sim is a similarity function, we considered cosine similarity.

$$\text{sim}(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{||\vec{x}||\,||\vec{y}||}$$

- p is the embedding of the query
- d is the embedding of the positive document
- n_i are the embeddings of the negative documents

This loss is computed for each document and then gets averaged on the batch to get the final result.

What the loss amounts to is similar to a cross entropy loss between query-positive and query-negative pairs for which the model must learn to maximize the similarity between query and positive while minimizing the ones between query and all negatives. This is reflected in embedding space by bringing the embeddings of positive pairs closer together while pushing the embeddings of negative pairs further apart. Training with several negative examples forces the model to learn to distinguish the positive pair from a variety of irrelevant sentences, leading to a more generalized and accurate ranking ability.

Another possible alternative is Symmetric MNRL, the mathematical formulation is similar but it also computes the inverse loss, that is:

1. Compute MNRL Loss with query - documents $\text{MNRL}(q, p, n_1, ..., n_m)$

2. Compute MNRL Loss with document - queries MNRL$(p, q, q_{n_1}, ..., q_{n_m})$
3. sum and minimize the total loss

Training was carried out using the built-in `trainer` from the `SentenceTransformer` package with specifications:

- `Batch size: 16`
- `Gradient accumulation: 2`
- `Optimizer: AdamW`
- `Learning rate: 5e^-6`
- `Eary Stopping Patience: 5 steps`
- `Best metric for early stopping: MAP@10`

The choice to use these parameters was made after preliminary experiments and taking into account the memory limitation of the system, which had specifications:

- `CPU: Intel Xeon E5-2680 v2 (10 Cores - 3.60 GHz)`
- `GPU: Nvidia GeForce RTX 3060Ti (16 GB)`
- `RAM: 128 GB`

Adam W[28] is the standard optimizer for the `SentenceTransformer` trainer and is an improved version of ADAM for weight decay. All trainings were performed with an early stopping callback based on the performance on the validation set. Once the training is completed the model are saved and evaluated on the test set Table 3.6.

MNRL Fine Tuning - Test Set				
Model Name	Loss Function	ACC@1	MAP@10	NDCG@10
MiniLM L12	-	0.424	0.526	0.576
MiniLM L12	MNRL	0.441	0.559	0.620
MiniLM L12	MNRL (Sym)	0.382	0.512	0.574
MPNET	-	0.456	0.550	0.602
MPNET	MNRL	0.499	0.611	0.665
MPNET	MNRL (Sym)	0.495	0.600	0.652
BGE	-	0.507	0.616	0.672
BGE	MNRL	0.618	0.733	0.782
BGE	MNRL (Sym)	0.574	0.696	0.750

Table 3.6: Fine Tuning - Test Set: MNRL shows better and more consistent results compared to symmetric MNRL. All models benefit from domain-specific fine tuning, with larger models experiencing a more noticeable increase in performance.

Results of fine-tuning show how:

– Standard MNRL consistently outperforms Symmetric MNRL for all models.

– Fine-tuning significantly boosts performance across all models, with a $+3.3\%$ increase in $MAP@10$ for MiniLM, $+6.1\%$ for MPNet, and $+11.7\%$ for BGE.

These results confirm that the generated questions are substantially out of distribution compared to the data the model was originally trained on. Another key observation is that larger models tend to benefit more from fine-tuning, likely due to their greater expressive capabilities.

Best Models

Regarding the final application in a RAG system it is useful to study the best models in cumulative accuracy. This mirrors more closely a RAG environement in which, assuming perfect context retention by the LLM, the question is correctly answered if the correct document is among the first k retrieved documents.

Expanded Metrics - Final Models							
Model Name	Fine Tuned	ACC@1	ACC@2	ACC@5	ACC@10	MAP@10	NDCG@10
SPLADE	-	0.181	0.235	0.343	0.388	0.244	0.279
MiniLM	no	0.424	0.531	0.672	0.733	0.526	0.576
MiniLM	yes	0.441	0.561	0.723	0.814	0.559	0.620
MPNet	no	0.456	0.559	0.678	0.765	0.550	0.602
MPNet	yes	0.499	0.631	0.765	0.834	0.611	0.665
BGE	no	0.507	0.623	0.770	0.851	0.616	0.672
BGE	yes	**0.618**	**0.770**	**0.881**	**0.934**	**0.733**	**0.782**
Cohere	-	0.552	0.674	0.795	0.872	0.658	0.710

Table 3.7: Retrieval Models Recap: all models significantly benefit from fine tuning, both in accuracy and in ranking performance. The larger BGE model is also able to surpass Cohere's performances in this configuration

It is particularly insightful to plot the difference in performance for each model between the base model and its fine-tuned counterpart (Figure 3.5). Notice how the larger models not only benefit more from fine-tuning but this improvement seems to show up for smaller values of k, in fact, the biggest difference between the base and ft-model is reached at $k = 10$ for MiniLM, $k = 5$ for MPNet and $k = 2$ for BGE.

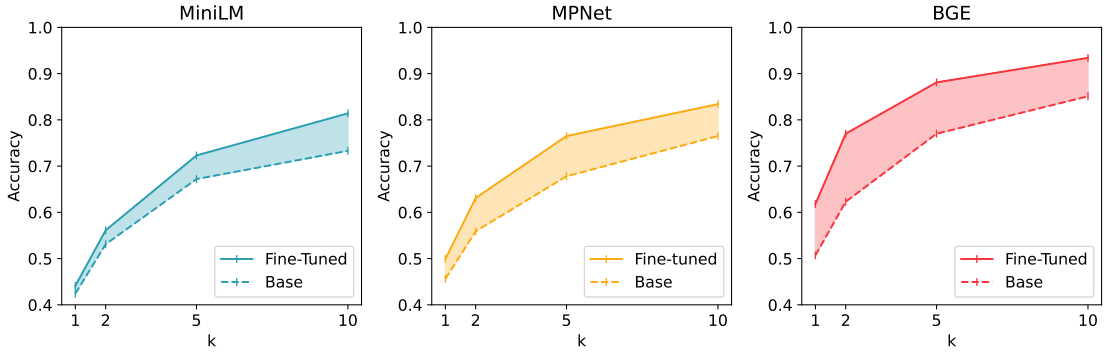


Figure 3.5: Effect of fine-Tuning on models: A larger band means greater difference between the base and fine-tuned models. Notice also how larger models (to the right) reach the maximum difference for smaller values of k

3.4 Full RAG Pipeline Evaluation

The last step for evaluating and quantifying the effectiveness of the fine-tuning in a real-world like scenario is setting up a full, End2End (E2E) RAG system on which the models will be tested and compared.

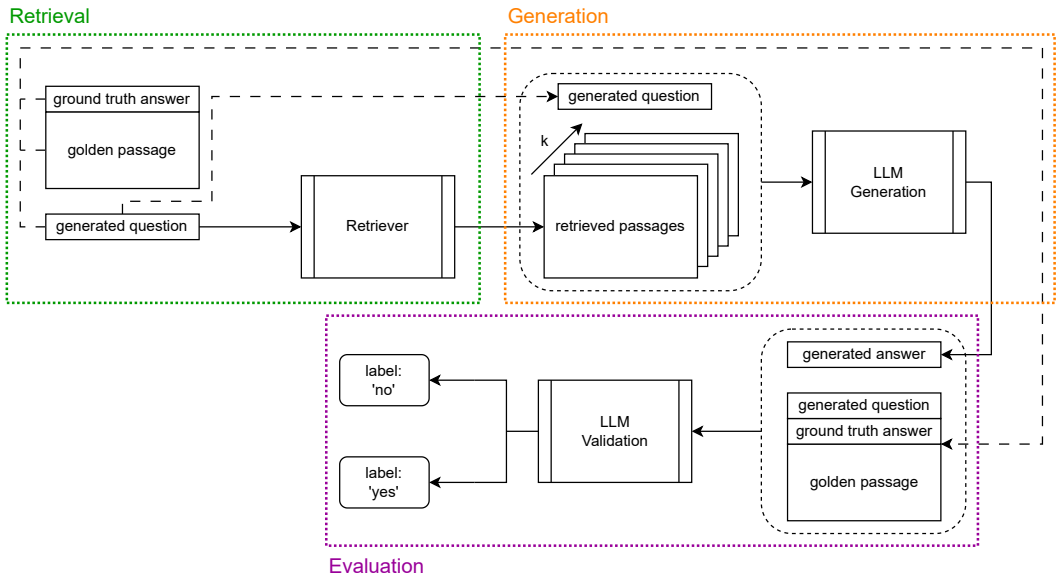


Figure 3.6: RAG Evaluation Pipeline: A three step process is set up to test and evaluate retrievers in a End2End scenario. The question is passed through the Retriever and the most relevant documents are passed to the LLM. The answer can then be Generated with additional context and Evaluated with an independent LLM.

The full pipeline is described in the diagram and can be thought as composed of three distinct section:

1. Retrieval: This is where the fine-tuned model comes into play. To establish a consistent baseline for all models, the top $k = 10$ documents were retrieved and provided as context to the LLM. This is used as a baseline to balance performance with latency and cost. Also, providing too much context to a LLM could result in a 'Lost in the Middle' problem [29] in which the LLM loses track of information in the middle of the prompt, resulting in diminished performance.
2. Generation: The generation component receives the query and the retrieved documents from the retriever and uses them to generate an answer. To optimize the prompt, self-refinement is employed. This process involves feeding a raw prompt into the selected LLM, allowing it to refine and optimize the prompt for its own use.
3. Evaluation: The final evaluation stage involves assessing the RAG system's answers using OpenAI's GPT-4 to ensure objectivity and minimize bias. The LLM checks if the RAG-generated answer matches the 'true' answer created in generation with full context. A rigorous set of guidelines was included in the prompt, and the model's temperature was set to $T = 0$ for consistency and stability.

This method provides a rigorous and impartial assessment of the RAG system's accuracy and reliability.

Results

All the models considered in the previous sections are evaluated End2End, with the addition of testing the QA capabilities of just the LLM with no additional context in order to test the base knowledge of the LLM and set a baseline for the QA task difficulty. In the case of `sentence-transformers`, both the base and fine-tuned version have been evaluated to measure the improvements. All results are found in Table 3.8 and, for the most significant models, the language specific performance is also explored in Table 3.9

Full RAG - Test Set		
Model Name	Fine Tuned	ACC
No RAG	-	0.461
BM-25+	-	0.439
SPLADE	-	0.542
MiniLM L12	no	0.765
MiniLM L12	yes	0.808
MPNET	no	0.780
MPNET	yes	0.821
BGE	no	0.834
BGE	yes	**0.878**
Cohere	-	0.832

Table 3.8: End2End RAG - Test Set: the answering accuracy is compared between different models to evaluate them and highlight the positive effects of fine tuning.

Regarding sparse models, BM-25+ performs worse than noRAG. This is most likely because, in a cross lingual setting, document search is no longer possible through keyword matching for alternative alphabets so the context returned to the LLM is mostly unrelevant and, as a consequence, can be confusing or misleading when producing an answer. This can be seen in the language-specific performance, with English and Italian being by far the best performers as the source languages of documents, while Japanese and Chinese show a sudden drop in performance. This, once again, shows the limits of purely lexical methods for cross-lingual information retrieval. SPLADE, on the other hand, is able to leverage the lexical expansion in order to improve performance but it still suffers on the asian alphabets. Regarding `sentence-transformers` models the benefits of fine tuning are noticeable, with MiniLM L12 gaining 4.3% in accuracy and performing just 2.4% worse than Cohere with only one third of the embedding dimension. MPNET sees a 4.1% improvement, 1.1% worse than Cohere. The larger BGE performs on par with Cohere even without Fine-Tuning and surpasses it with fine tuning with a $+4.4\%$ on accuracy. The language performance is much more consistent compared to lexical models, confirming the powerful expressive capabilities of the models.

Language Evaluation									
Model Name	de	en	es	fr	it	ja	vi	zh	AVG
noRAG	0.574	0.464	0.431	0.500	0.378	0.400	0.490	0.453	0.461
BM-25	0.404	0.595	0.451	0.464	0.689	0.271	0.431	0.293	0.439
SPLADE	0.489	0.726	0.667	0.607	0.600	0.400	0.373	0.427	0.542
MiniLM	0.787	0.845	0.784	0.786	0.867	0.833	0.824	0.747	0.808
MPNET	0.787	0.893	0.843	0.804	0.911	0.700	**0.863**	0.773	0.821
BGE	**0.936**	**0.940**	**0.882**	**0.839**	**0.933**	0.850	0.843	**0.813**	**0.878**
Cohere	0.809	0.917	0.843	0.750	0.844	**0.867**	0.824	0.773	0.832

Table 3.9: End2End RAG - Language Performance: lexical based models perform less reliably from language to language, and notably even fail when question and documents are not in the same alphabet. Dense models show better and more consistent performance.

3.4.1 Experimenting with context length

Full RAG - Test Set		
Model Name	k	ACC
BGE	5	0.849
BGE	10	0.878
BGE	15	0.904
BGE	20	0.915

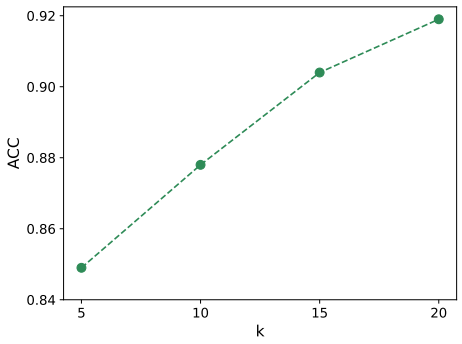


Figure 3.7: End2End RAG - Context Length: increasing the number of passages retrieved and passed to the LLM has a positive effect on accuracy, highlighting the long-context performance of the LLM.

It can be interesting to study the impact of context length to accuracy. During all previous tests the top $k = 10$ documents were retrieved, now the number of documents will be varied. Due to cost and time constraints, only the best performing model has been tested and results are shown in Table 3.7. The accuracy improves as the number of retrieved documents increases, indicating that the LLM is well capable to process and utilize longer contexts. This suggests that providing the model with more context allows it to generate more accurate and relevant responses. However, increasing the context also means that the input prompts become longer, which leads to higher latency and increased operational costs due to the computational resources required to process the larger inputs. Therefore there is a need to carefully balance the benefits of accuracy with the trade-offs in latency and cost, that will depend on the specific use case scenario.

3.5 Complex questions

The next important section of the study regards complex questions. Complex questions are, generally speaking, questions that require multiple documents to be answered [30]. We want to extend the general framework introduced in the previous section for this specific use case, generating complex questions spanning multiple documents to be later used for fine-tuning sentence transformer models.

Complex Question - Example

An example of a complex question is

```
'Are lions bigger than fridges?'
```

The implicit chain of thought reasoning needed to answer such question is

```
How big is a lion?
How big is a fridge?
Which size is larger?
```

The likelihood of finding both answers in a single documents is extremely low so a multi document search is necessary

3.5.1 Generation

The generation procedure is extremely similar to the one highlighted in the previous section, the only difference is that now the LLM is tasked with providing a question that encompasses multiple documents. In this study we considered two documents questions. The general procedure is

Complex Question Generation PseudoCode

```
#GENERATION
- Select document 1
- Select document 2
- Read the documents and create a question that encompasses both.

#VALIDATION
- Answer the question in a separate call given full context
- If the generation is successful add the QA pair to the dataset, otherwise
discard
```

The Train-Test split is the same as the previous section and the final dataset features are shown in Tables 3.10 and 3.11.

	Documents	Passages	Questions
Train	51	442	2139
Validation	6	75	357
Test	7	96	471

Table 3.10: Train/Test Split - Complex Questions

Language Split - Complex Questions			
Language	Questions	Language	Questions
de	385	it	361
en	311	ja	387
es	383	vi	391
fr	386	zh	363

Table 3.11: Language Split - Complex Questions

3.5.2 Fine-tuning the embedder

A comprehensive evaluation procedure has been set up for the complex questions dataset. The main difference is that now each query has two positive samples, being the two documents from which it originates.

1. MNRL on complex questions: the training procedure is equivalent to the one carried out for standard questions but this time CQ are used

$$CQ \rightarrow \begin{matrix} \text{MNRL}(CQ, p_1, n) \\ \text{MNRL}(CQ, p_2, n) \end{matrix}$$

2. MNRL on split questions: complex questions are first split into simple sub-questions using an LLM, which are then used to MNRL train the embedder

$$CQ \rightarrow (Q_1, Q_2) \rightarrow \begin{matrix} \text{MNRL}(Q_1, p_1, n) \\ \text{MNRL}(Q_2, p_2, n) \end{matrix}$$

3. Custom loss MNRL+MSE con complex questions: In this custom loss a ranking term in the form of MNRL loss has been combined with a MSE term on the embeddings between the complex question and each split question. This is done to try, on top of correcly ranking documents, also to bring closer the embedding of the split sub-questions to the embedding of the original one

$$CQ \rightarrow \begin{matrix} \text{MNRL}(CQ, p_1, n) + \text{MSE}(\text{CQ}, \text{Q}_1) + \text{MSE}(\text{CQ}, \text{Q}_2) \\ \text{MNRL}(CQ, p_2, n) + \text{MSE}(\text{CQ}, \text{Q}_1) + \text{MSE}(\text{CQ}, \text{Q}_2) \end{matrix}$$

4. Custom loss MNRL+MSE con split questions: the idea is very similar to the previous loss but MNRL is

performed on the split questions.

$$CQ \rightarrow (Q_1, Q_2) \rightarrow \frac{1}{2}(\text{MNRL}(Q_1, p_1, n) + \text{MNRL}(Q_2, p_2, n)) + \text{MSE}(\text{Q}_1, \text{CQ}) + \text{MSE}(\text{Q}_2, \text{CQ})$$

5. Multi task training on complex questions: another way in which the two information can be combined is through a multi-task training, in which two losses on two different datasets are jointly minimized. In this first case with complex questions

$$CQ \rightarrow \text{task 1:} \begin{matrix} \text{MNRL}(CQ, p_1, n) \\ \text{MNRL}(CQ, p_2, n) \end{matrix} \quad \text{task 2:} \begin{matrix} \text{MSE}(\text{CQ}, \text{Q}_1) + \text{MSE}(\text{CQ}, \text{Q}_2) \\ \text{MSE}(\text{CQ}, \text{Q}_1) + \text{MSE}(\text{CQ}, \text{Q}_2) \end{matrix}$$

6. Multi task training on split questions: The training procedure is largely similar to the previous case but the MNRL training is performed on split questions

$$CQ \rightarrow (Q_1, Q_2) \rightarrow \text{task 1:} \begin{matrix} \text{MNRL}(Q_1, p_1, n) \\ \text{MNRL}(Q_2, p_2, n) \end{matrix} \quad \text{task 2:} \begin{matrix} \text{MSE}(\text{CQ}, \text{Q}_1) + \text{MSE}(\text{CQ}, \text{Q}_2) \\ \text{MSE}(\text{CQ}, \text{Q}_1) + \text{MSE}(\text{CQ}, \text{Q}_2) \end{matrix}$$

All these training procedures have been performed on the MPNet model with variable number of hard negatives (0 to 3 hard negatives). An alternative batch sampler has been used to ensure no multiple positive examples in any given batch. All results can be found in the appendix but, for clarity, only the best model for each training type will be presented in Table 3.12.

Multi Modal Training - Test Set					
Training Type	Hard Negatives	ACC@1	ACC@2	MAP@10	NDCG@10
No Training	-	0.437	0.588	0.366	0.468
Simple Questions	-	0.453	0.615	0.372	0.484
1	0	0.414	0.556	0.333	0.435
2	1	**0.475**	0.615	**0.382**	0.486
3	1	0.427	0.571	0.357	0.460
4	0	0.473	**0.618**	0.381	**0.491**
5	0	0.463	0.609	0.366	0.472
6	0	0.466	0.606	0.380	0.485

Table 3.12: Complex Questions - Fine Tuning: performance depends primarily on the type of questions used for MNRL loss, with little influence given by MSE terms and Hard Negatives

Training directly on complex questions (training number 1 and 3) provides unsatisfactory results, performing worse than the base model. This could be explained by the fact that the complex question embedding is close to neither single document and trying to get both documents closer to the query at the same time can be confusing for the trainer. The best pretrained model from the previous section improves results, likely because of the shared document pool between simple and complex questions. Trainings on split questions perform the best on average, with very similar performance between the three procedures. Of the three, the simple MNRL is not only the best performing but also the one requiring the simpler dataset, loss function, and computational load, so it is

considered to be the best model. As it was observed in the previous section Hard Negatives have little to no influence in performance.

In order to make the process less time consuming, taking MPNet as a benchmark, BGE has been fine-tuned only using method 2, and the results are presented in Table 3.13. The results from simple questions are reflected even in this case, with the larger model showing a more consistent improvement across all ranking metrics.

Complex questions - Test Set				
Model Name	Fine-Tuned	ACC@1	MAP@10	NDCG@10
MPNet	no	0.437	0.366	0.468
MPNet	yes	0.475	0.382	0.486
BGE	no	0.510	0.421	0.534
BGE	yes	0.597	0.475	0.581
Cohere	-	0.556	0.467	0.572

Table 3.13: Final Models - Test Set: the improvements in performance are observed even for the larger model, which now surpasses Cohere in retrieval accuracy and precision

Due to the added complexity of the task the performance difference between simpler and more complex models is more noticeable in this case.

3.5.3 Retrieval Strategies

When handling complex queries there are multiple ways in which retrieval can be performed. We will present three progressively more complex retrieval methods, highlighting strengths and weaknesses for each.

Direct retrieval: In this mode the retrieval is the same as the standard RAG, in which the complex question is fed to the retriever and the answer is generated based on the retrieved documents and the complex question. in this framework, for each search:

```
n_retrievals = 1
n_generations = 1
```

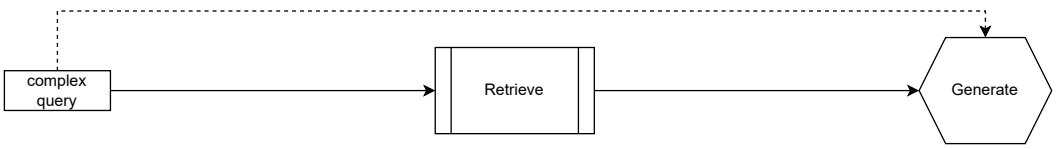


Figure 3.8: Standard RAG: the complex question is used directly to retrieve the most relevant documents

The main advantage is the ease of use since it does not require any adaptation compared to the standard RAG, also the number of retrievals and generations is kept at a minimum, ensuring the lowest latency and operational costs.

Multi stage retrieval: The complex question is first divided into n simple sub-questions with an LLM call, retrieval is performed on each split question separately and the top scoring documents globally are used to answer the question

```
n_retrievals = n
n_generations = 2
```

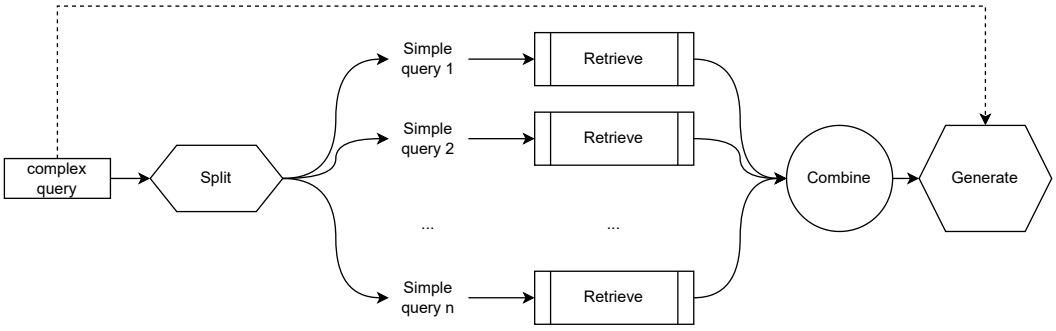


Figure 3.9: Question decomposition with multiple retrieval: the complex question is first split into simpler sub questions, which are separately used for retrieval.

Based on the fine-tuning results this solution wants to leverage the added expressive capabilities of sub-questions while keeping the number of generations as low as possible.

Multi generation retrieval: The complex question is first divided into n simple sub-questions with an LLM, retrieval is performed on each split question separately and an answer is produced for each sub question, then all answers are combined to form the final answer

```
n_retrievals = n
n_generations = 2+n
```

This is by far the most comprehensive and complex retrieval technique.

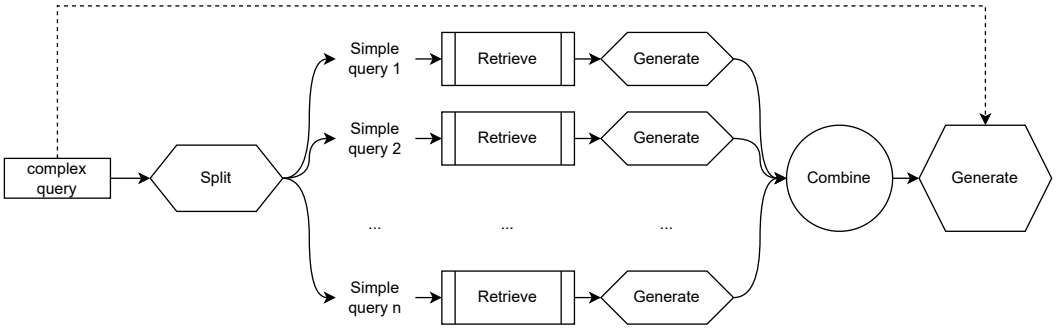


Figure 3.10: Question decomposition with multiple generations: the question is split and retrieval+generation is performed for every sub-question separately. Then, the separate sub-answers are combined into a single answer.

3.5.4 End2End Evaluation

As for the simple questions case the final evaluation is performed in a End2End fashion with a Claude generation and a GPT evaluation. Along with the different architectures the retrieval techniques have also been explored (Table 3.14).

Full RAG - Test Set			
Model Name	Fine Tuned	Retrieval Type	ACC
No RAG	-	Direct	0.711
MPNET	no	Direct	0.741
MPNET	yes	Direct	0.745
MPNET	yes	Multi Ret	0.786
MPNET	yes	Multi R+G	0.836
BGE	no	Direct	0.816
BGE	yes	Direct	0.819
BGE	yes	Multi Ret	0.827
BGE	yes	Multi R+G	0.853
Cohere	-	Direct	0.799
Cohere	-	Multi Ret	0.799
Cohere	-	Multi R+G	0.864

Table 3.14: End2End RAG - Complex Questions: Fine-tuned models benefit from multi retrieval, unlike Cohere. On multi generation+retrieval all models have a noticeable increase in performance.

NoRAG performs much better compared to the test on simple questions, probably because generating coherent but multi-hop questions led to more constraint in generation, which resulted in an inherently easier task. `sentence-transformer` models are only marginally affected by fine-tuning in direct retrieval ($+0.3\%$ for both MPNet and BGE) while the improvement is much more noticeable in Multi Retrieval ($+4.1\%$ for MPNet and $+0.8\%$ for BGE), this difference can be explained by the fact that models have been fine-tuned on split questions and thus benefit more from split retrieval, for the same reason it is reasonable not to observe any improvement for Cohere between the two retrieval techniques. Multi Retrieval+Generation is, by far, the best performing retrieval pipeline, with $+9.1\%$ for MPNet, $+3.7\%$ for BGE and $+6.5\%$ for Cohere, the latter being the best model overall.

Latency studies

On top of differences in accuracy it is extremely relevant to study also the mean inference times of the different QA techniques described above to assess their usability in the real world. Increased latency in answering must be accompanied by substantial and relevant improvements in accuracy to compensate for the degraded user experience

resulting from a slower QA system. The two main contributors in latency for RAG systems are the retriever and the LLM. The retriever has to embed the query (embedding latency) and then perform the retrieval by computing and ranking cosine similarities between the query and all documents (retrieval latency). Embedding latency depends on the hardware and the model size if the embedding is computed locally or on the API retrieval time if the embedding is done through cloud computing. Retrieval latency depends, other than on local hardware, on the embedding dimension and size of the document pool. For the models considered in this thesis the latency induced by the API call is noticeably greater than the local computing time (Table 3.15)

Model	Embedding+Retrieval Latency
MiniLM	0.057 ± 0.004s
MPNet	0.060 ± 0.005s
BGE	0.090 ± 0.007s
Cohere	0.37 ± 0.03s

Table 3.15: Retrieval Latency: Latency is much more affected by the API call time in Cohere rather than the local inference time. (Average and Standard deviation on 100 calls)

Latency on the LLM largely depends, on top of the API call latency, on the length of the input prompt and the amount of output text, with longer inputs and outputs requiring more computational time. An estimation for the ideal API latency time has been computed by using a very short prompt and resulted in 0.8 ± 0.3s while, for a RAG call processing time resulted in 7.6 ± 1.8s. Another element to note is how, both for `Cohere` and `Claude` the first call is slightly slower than subsequent calls due to the lazy nature of the API. Considering that, ideally, a high quality chat-bot should provide a correct answer in one or very few shots this added latency needs to be taken into consideration.

In this current implementation the biggest contributor to latency is the LLM call, so the retrieval technique largely determines latency, with more complex options like multi-generation being heavily penalized. Using locally run sentence transformer models still has notable impact by taking negligible time in embedding compared to the latency of a Cohere API call.

3.6 Ablation Studies

3.6.1 Ablation-Hard Negatives MNRL Training

Hard negatives represent a potentially valuable resource in training for contrastive learning. These are documents that, while unrelated to the positive document, are close to it in feature space and thus semantically similar. By providing more nuanced information to the sentence embedder during training, hard negatives can enhance the model's discriminative capabilities. It's worth noting that when incorporating hard negatives, symmetric MNRL is no longer a viable option, as the inverse loss computation loses its significance.

Hard Negatives Mining

Hard negatives mining is the process of extracting hard negatives for a dataset. Hard negatives mining is performed only once on a dataset using a high peformance model, in our case Cohere. The pipeline was implemented as follows

1. Embed all documents and queries
2. Compute, query by query, the similarity between query and all documents
3. Select the top n documents by score for each query
4. Remove, if present, the correct document
5. Consider the remaining $n - 1$ documents as the hard-negatives and create a database

For this research the top $n = 20$ hardnegatives have been saved. Once all HN have beeen computed, it is important to decide how to present them during MNRL training. For this three different techniques have been explored.

Parallel HN: In the dataset the top n hard negatives are fed in parallel for each positive sample, the row of the dataset has shape

```
(query, positive, negative_1, ..., negative_n)
```

It is worth noting that adding hard negatives in this way makes the dataset progressively heavier to store and process during training. The results are

Top-k Sampling: Only one hard negative is fed for each positive, but the HN is randomly sampled from the top-k hard negatives at each iteration

```
(query, positive, random_negative)
```

Threshold Sampling: The hard negative is sampled from the ones with a maximum score difference from the top HN

```
(query, positive, threshold_negative)
```

MiniLM - Deterministic HN		
hard negatives	ACC@1	MAP@10
1	0.496	0.631
2	0.501	0.633
3	0.521	0.644
4	0.521	0.639
5	0.513	0.635

Table 3.16: MiniLM - Deterministic HN

MPNET - Deterministic HN		
hard negatives	ACC@1	MAP@10
1	0.560	0.695
2	0.574	0.693
3	0.577	0.696
4	0.591	0.700
5	0.579	0.697

Table 3.17: MPNet - Deterministic HN

MiniLM - Sampled HN		
Sample pool	ACC@1	MAP@10
5	0.513	0.639
10	0.504	0.631
15	0.504	0.638
20	0.513	0.635

Table 3.18: MiniLM - Sampling HN

MPNET - Sampled HN		
Sample pool	ACC@1	MAP@10
5	0.593	0.706
10	0.599	0.709
15	0.582	0.698
20	0.596	0.705

Table 3.19: MPNet - Sampling HN

Depending on the threshold the number of hard negatives available to each sample changes, but a suitable indicator can be the average number of negatives.

Due to memory constraints in the system, integrating hard negatives into the BGE model was not feasible for this particular study, so it has not been carried out, but would be a notable starting point for future research.

Based on the results provided by the validation set we can notice:

- MiniLM: adding hard negatives does not provide any improvement, likely due to the reduced expressive power of the model, that cannot benefit from the added context given by hard negatives
- MPNet: adding hard negatives significantly improves the performance of the model. The various sampling techniques vary in effectiveness with the best one being the random sample from the top 10 mined hard negatives.

Evaluating those models on the Test Set, however, shows diminished performances compared to the standard fine-tuning, shown in Table3.22. This could likey be a sign of overfitting on the Validation set during model selection. Adding additional fine-grained information, while adding expressive capabilities to the model, ultimately led to a less general representation that loses performance on unseen data. For this reason, the standard MNRL trained models are kept as the best models.

MiniLM - Threshold HN			
Threshold	Avg Hard Neg	ACC@1	MAP@10
0	1	0.496	0.631
0.025	3.0	0.513	0.638
0.05	7.8	0.524	0.640
0.075	13.5	0.507	0.635
0.1	17.3	0.515	0.635

Table 3.20: MiniLM - Threshold HN

MPNET - Threshold HN			
Threshold	Avg Hard Neg	ACC@1	MAP@10
0	1	0.560	0.695
0.025	3.0	0.571	0.695
0.05	7.8	0.579	0.703
0.075	13.5	0.588	0.696
0.1	17.3	0.577	0.697

Table 3.21: MPNet - Threshold HN

MNRL Fine Tuning + HN - Test Set				
Model Name	Loss Function	ACC@1	MAP@10	NDCG@10
MiniLM L12	MNRL	0.441	0.559	0.620
MiniLM L12	MNRL + HN	0.388	0.502	0.557
MPNET	MNRL	0.499	0.611	0.665
MPNET	MNRL + HN	0.463	0.587	0.645

Table 3.22: Hard Negative Fine Tuning - Test Set

4
Conclusions

In this thesis the impact of artificial data generation for domain specific RAG application has been implemented, tested and validated. The comprehensive generation plus fine-tuning pipeline lead to a significant increase in performance across a wide variety of models and application, even surpassing current SOTA solutions in multiple benchmarks (up to $+4.4\%$ in End2End accuracy). The possibility to readily deploy the framework locally on resource constrained hardware makes it a compelling solution for a wide variety of business applications for which, after an initial upfront investment for generating the synthetic dataset and fine-tuning, the models' running costs are greatly diminished.

Adapting the approach to complex questions has led to improvements in retrieval with and without changing the retrieval techniques, with the best results for fine-tuning obtained in multi retrieval generation ($+4.4\%$ and $+1.1\%$ for MPNet e BGE respectively). On the other hand the process has room for improvements and would probably benefit from a more nuanced generation procedure which ensures the complexity of the generated question while still finding an answer. Various ideas for further research are proposed and will likely be explored in future works in order to further increasing accessibility and lowering operational costs:

- Train models with Matryoshka embeddings [31], in order to provide highly informative compressed embeddings for faster and lighter retrieval.
- Use a smaller LLM both for RAG and question splitting in the case of complex questions during operation. While generating realistic QA pairs requires SOTA models the everyday operations can be scaled-down to the specific needs, thus lowering costs and possibly allow models to be run locally, adding another layer of safety to private or sensitive data.

After having established the validity of the method from a research standpoint the natural next step would be to create a comprehensive and deployable pipeline which, given a document, it is able to:

1. Extract text

2. Generate QA pairs
3. Fine Tune retrieval models

From there, the new retriever could be integrated into already existing solutions such as virtual assistants, Chatbots or even implemented into agents to perform field specific tasks, thus enabling tailored solutions for user-specific applications.

References

[1] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 9459–9474. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 11 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[3] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," 2014. [Online]. Available: https://arxiv.org/abs/1406.1078

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[5] D. M. W. Powers, "Applications and explanations of Zipf's law," in *New Methods in Language Processing and Computational Natural Language Learning*, 1998. [Online]. Available: https://aclanthology.org/W98-1218/

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019. [Online]. Available: https://arxiv.org/abs/1810.04805

[7] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[9] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," 2016. [Online]. Available: https://arxiv.org/abs/1609.08144

[10] T. Capes, P. Coles, A. Conkie, L. Golipour, A. Hadjitarkhani, Q. Hu, N. Huddleston, M. Hunt, J. Li, M. Neeracher, K. Prahallad, T. Raitio, R. Rasipuram, G. Townsend, B. Williamson, D. Winarsky, Z. Wu,

and H. Zhang, "Siri on-device deep learning-guided unit selection text-to-speech system," in *Interspeech 2017*, 2017, pp. 4011–4015.

[11] D. Driess, F. Xia, M. S. M. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, W. Huang, Y. Chebotar, P. Sermanet, D. Duckworth, S. Levine, V. Vanhoucke, K. Hausman, M. Toussaint, K. Greff, A. Zeng, I. Mordatch, and P. Florence, "Palm-e: An embodied multimodal language model," 2023. [Online]. Available: https://arxiv.org/abs/2303.03378

[12] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," 2023. [Online]. Available: https://arxiv.org/abs/1910.10683

[13] N. Maslej, L. Fattorini, R. Perrault, V. Parli, A. Reuel, E. Brynjolfsson, J. Etchemendy, K. Ligett, T. Lyons, J. Manyika, J. C. Niebles, Y. Shoham, R. Wald, and J. Clark, "The ai index 2024 annual report," AI Index Steering Committee, Institute for Human-Centered AI, Stanford University, Stanford, CA, Technical Report, April 2024.

[14] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[15] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn, "Direct preference optimization: Your language model is secretly a reward model," 2024. [Online]. Available: https://arxiv.org/abs/2305.18290

[16] Amazon bedrock. [Online]. Available: https://aws.amazon.com/bedrock/

[17] D. Laredo, C. Cortes, G. Velazquez, , and S. Córdova. Prompt engineering techniques and best practices: Learn by doing with anthropic's claude 3 on amazon bedrock. [Online]. Available: https://aws.amazon.com/blogs/machine-learning/prompt-engineering-techniques-and-best-practices-learn-by-doing-with-anthropics-claude-3-on-amazon-bedrock/

[18] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W. tau Yih, "Dense passage retrieval for open-domain question answering," 2020. [Online]. Available: https://arxiv.org/abs/2004.04906

[19] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. [Online]. Available: http://arxiv.org/abs/1908.10084

[20] T. Formal, B. Piwowarski, and S. Clinchant, "Splade: Sparse lexical and expansion model for first stage ranking," 2021. [Online]. Available: https://arxiv.org/abs/2107.05720

[21] Claude 3.5 sonnet. [Online]. Available: https://www.anthropic.com/news/claude-3-5-sonnet

[22] A. Trotman, A. Puurula, and B. Burgess, "Improvements to bm25 and language models examined," in *Proceedings of the 19th Australasian Document Computing Symposium*, ser. ADCS '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 58–65. [Online]. Available: https://doi.org/10.1145/2682862.2682863

[23] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, "Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers," 2020. [Online]. Available: https://arxiv.org/abs/2002.10957

[24] A. Conneau, K. Khandelwal, N. Goyal, V. Chaudhary, G. Wenzek, F. Guzmán, E. Grave, M. Ott, L. Zettlemoyer, and V. Stoyanov, "Unsupervised cross-lingual representation learning at scale," 2020. [Online]. Available: https://arxiv.org/abs/1911.02116

[25] J. Chen, S. Xiao, P. Zhang, K. Luo, D. Lian, and Z. Liu, "Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation," 2024.

[26] Introducing embed v3. [Online]. Available: https://cohere.com/blog/introducing-embed-v3

[27] M. Henderson, R. Al-Rfou, B. Strope, Y. hsuan Sung, L. Lukacs, R. Guo, S. Kumar, B. Miklos, and R. Kurzweil, "Efficient natural language response suggestion for smart reply," 2017.

[28] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," 2019. [Online]. Available: https://arxiv.org/abs/1711.05101

[29] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, "Lost in the middle: How language models use long contexts," 2023. [Online]. Available: https://arxiv.org/abs/2307.03172

[30] M. Gabburo, N. P. Jedema, S. Garg, L. F. R. Ribeiro, and A. Moschitti, "Measuring retrieval complexity in question answering systems," 2024. [Online]. Available: https://arxiv.org/abs/2406.03592

[31] A. Kusupati, G. Bhatt, A. Rege, M. Wallingford, A. Sinha, V. Ramanujan, W. Howard-Snyder, K. Chen, S. Kakade, P. Jain, and A. Farhadi, "Matryoshka representation learning," 2024. [Online]. Available: https://arxiv.org/abs/2205.13147

A

Appendix

Question Generation

Question Generation

```
import random

def langselect():
    languages = ['Italian', 'English', 'Spanish', 'German', 'Japanese',
                 'French', 'Chinese', 'Vietnamese']

    return(random.choice(languages))

def buildprompt(data, lang = None, nquestions = 1):
    questionspec = ''
    for i in range(nquestions):
        if lang==None:
            language = langselect()
        else:
            language = lang

        questionspec = questionspec + f'generate question number {i+1}
        in language {language}, '
```

```
    doc_cont = data["content"]
    doc_name = data['doc_name']
    header = data['header']

    docspec = f'Document name : {doc_name}, header name : {header},
    Document content : {doc_cont}'

messages = [
    (
    "system",

    #task context
    """You are an expert AI assistant in artificial data generation,
    specializing in question creation for data augmentation in information
    retrieval. You will receive excerpts from business documents along with
    their titles and need to create human-like questions that target specific,
    informative parts of the excerpts.""" +

    #Reasoning
    """
    Take your time in understanding the meaning of the whole excerpt before
    thinking about meaningful questions.
    """ +

    #Detailed task Description and Rules
    """
    If you are asked to generate a question in a language different from
    the document consider translating the document first before creating
    the question.
    Here are some important rules for generating correcly formatted questions:
    - Questions should be conversational and informal in tone,
    strongly avoiding lengthy or overly specific phrasing.
    - They can be formulated either as questions or as assertive instructions
    - Avoid more than one sentence per question,
    try to make a question shorter that 10 words.
    - Avoid referencing specific parts of the document in the question
    - Questions should be focused on open-ended information
    (not yes/no or confirmatory questions).

    REMEMBER: generate questions based only on the information contained
```

```
    in the excerpt, do not rely on any previous knowledge.
    REMEMBER: if you are not able to generate any meaningful question,
    return 'NA' """ +

    #In-Context Learning
    """Some examples for the generated questions are:
    <example>
    qual è la tariffa gionaliera di rimborso spese di omnys per le trasferte?
    mi dai il numero di davide pozza?
    di che cosa si occupa filmop?
    </example>
    #These questions are human made question recorded during the test run
    of the internal company chatbot "Omnyscent"
    """
        ,
    ),

    (
"human",

#Task instructions
f"""Select specific informative segments in {docspec} and generate {nquestions}
different questions in the user-specified languages {questionspec}.
If the excerpt lacks enough detail to form meaningful questions,
inform the user without generating a question.""" +

#Format specifications
"""Return only the question without any other surrounding text in a
Python dictionary-like format where every question is formatted as
question:generated question, language:2 letter language abbreviation lowercase.
if you have to generate more than one question then the separator
between dictionaries should be a double newline """ +

# In-context Learning
One example for the output schema is
{"question": "What are the key responsibilities of the
digital preservation security manager?",
"language": "en"}"""
 ),
 ]
```

```
return messages
```

Question Validation

Question Validation

```
def buildpromptanswer(documents):

    doc_cont = [documents['content']]
    doc_header = list(documents['header'])
    doc_title = list(documents['doc_name'])
    question = documents['question']
    language = documents['language']


    fill = """Return only the answer without any other surrounding text in a
    Python dictionary like format where every question is formatted as
              question:generated question. One example for the output schema is
              {"answer": "This is a sample answer"}
              IMPORTANT: do not add any text before or after he dictionary"""

    documentspec = ''

    for i in range(len(doc_cont)):
        documentspec = documentspec + f'Document {i+1} title: {doc_title[i]}.
        Document {i+1} section name: {doc_header[i]}. Document {i+1} content:
        {doc_cont[i]}'

    messages = [
        (
        "system",
        """
        You are an expert and helpful AI assistant employed in a RAG task.
        You will receive a question to answer and a relevant document in which
        you will find the answer.
        Take your time in understanding the meaning of the question and the
        content of the document before producing the answer.
        The question and the document can possibly be in different languages.
        Consider translating the document in the same
```

```
        language as the question before providing an answer

        Here are some important rules for generating correcly formatted questions:
        - Answers should be clear and accurate
        - Do not provide any extra flavor text beyond the necessary
        - Keep answers as short as possible

        IMPORTANT: the answer must be provided in the same language as the
                question
        IMPORTANT: the questions were created based on the content of the
        documents so you should find the answer
        IMPORTANT: if you think the question is malicious or could compromise
        the safety of the company give as an answer 'denied_safety'
        IMPORTANT: if you can't find the answer in the documents give as an
        answer 'NA'
        """
        ),
        ("human", f"""Answer this question: {question} in the language
        {language} given these passages as context {documentspec}, provide
        a clear and concise answer.
            Use only the content of the documents to generate the answers
            and do not rely on your previous knowledge. {fill}"""
        ),
        ]

    return messages
```

RAG-Generation

Generation Prompt

```
def buildpromptanswer(documents, lang = None):

    doc_cont = list(documents['document_content'])
    doc_header = list(documents['document_header'])
    doc_title = list(documents['document_title'])
    question = documents['question']
    language = documents['language']
```

```
documentspec = ''

for i in range(len(doc_cont)):
    documentspec = documentspec + f'Document {i+1} title: {doc_title[i]}.
                    Document {i+1} section name: {doc_header[i]}.
                    Document {i+1} content: {doc_cont[i]}'

messages = [
    ("system",
    f'''
    You are an expert multilingual AI assistant in the final stage of a
    cross-lingual Retrieval-Augmented Generation (RAG) task. Your role
    is to provide accurate and concise answers based on the given question
    and relevant documents, even when they are in different languages.

    Input:
    - question: {question}
    - documents: {documentspec}
    - question_language: {language}

    Task:
    1. Analyze the question carefully.
    2. Review all provided documents, regardless of their language.
    3. Synthesize information from the documents to formulate an
       accurate answer.
    4. Provide the answer in the same language as the input question.

    Guidelines for answer generation:
    - Ensure answers are clear, accurate, and directly address the question.
    - Keep answers concise without sacrificing essential information.
    - Use only the provided documents to construct your answer.
    - If the documents are in a different language from the question,
      translate the relevant information accurately.
    - Maintain a conversational tone in your response.
    - Do not use lists or bullet points in your answer.
    - Avoid using line breaks within the answer.
    - Always attempt to provide an answer, even if you're unsure.
      Use qualifying language if necessary.

    Output format:
```

```
        Provide your response as a valid JSON object with the following structure:

        {
        "gen_answer": "Your generated answer here",
        "language": "language_code"
        }

        Example input:
        {
        "question": "Quels sont les principaux attraits touristiques de Paris?",
        "documents": [
            "Paris, the capital of France, is known for its iconic landmarks...",
            "Le Louvre, situé au cœur de Paris...",
            "Notre-Dame de Paris, eine berühmte mittelalterliche Kathedrale..."
        ]
        }

        Example output:
        {
        "gen_answer": "Les principaux attraits touristiques de Paris...",
        "language": "fr"
        }

        Remember to prioritize accuracy and relevance in your responses while
        adhering to the specified format and guidelines. Always strive to
        provide an answer, using the information available in the provided
        documents, and ensure you correctly infer and match the language of
        the input question.
        ''')
        ,
        ("human", f"""Answer this question: {question}"""),
        ]

    return messages
```

Answer Evaluation

```
def buildpromptanswereval(documents):
    doc_cont = [documents['document_content']]
    doc_header = [documents['header']]
    doc_title = [documents['document_name']]
    question = documents['question']
    genans = documents['gen_answer']
    refans = documents['answer']

    documentspec = ''.join(
        [f'Document {i+1} title: {doc_title[i]}. Document {i+1} section name:
        {doc_header[i]}. Document {i+1} content:
        {doc_cont[i]}' for i in range(len(doc_cont))])

    fill = """Return only the answer without any other surrounding text in a
            Python dictionary like format. One example for the output schema is
            {"similarity": "equivalent"}"""

    messages = [
            {"role": "system",
            "content" : f'''
            You are an expert AI assistant, specialized in evaluating responses
            generated by RAG systems. Your task is to assess the correctness
            of a generated answer by comparing it with a reference answer and
            the provided documents.

            Input:
            question: {question}
            reference document: {documentspec}
            reference answer: {refans}
            generated answer: {genans}

            Evaluation Rules:
            1. If the generated answer is empty, the evaluation must be 'no'.
            2. Use both the reference answer and the reference document to
               accurately assess the correctness of the generated answer.
            3. Focus on the semantic meaning of the answer rather than on
               specific terms used.
```

```
            4. The generated answer must be factually correct compared to
               the reference answer and the reference document.

            Expected Output:
            - Respond 'yes' if the generated answer is correct and consistent
               with the reference answer and reference document.
            - Respond 'no' if the generated answer is incorrect, inconsistent,
              or lacks crucial information present in the reference answer or
              reference document.

            Evaluation Process:
            1. Carefully read the question, reference document,
               and reference answer.
            2. Analyze the generated answer by comparing it with the reference
               answer and the information in the reference document.
            3. Assess whether the generated answer accurately captures the key
               information and meaning of the reference answer.
            4. Verify that there are no factual errors or inconsistencies with
               respect to the reference document.
            5. Determine if the generated answer adequately addresses the
               question posed.

            Provide your final evaluation ('yes' or 'no') based on this
            thorough analysis.
            '''
            + '''
            Return only the answer without any other surrounding text in a
            Python dictionary like format. One example for the output schema
                {"similarity": "yes"}
            '''
        },
        {
            "role": "user",
            "content": f"""Given the generated answer {genans} and the correct
            reference answer {refans}, tell me whether the two answers are
            similar or not according to your internal grading scale."""
        },]

    return messages
```

Fine Tuning Complex Questions

Results of fine tuning sentence transformers on the validation dataset, used for model selection.

MPNet - Validation Set				
Training type	Hard Negatives	ACC@1	MAP@10	NDCG@10
no	-	0.266	0.383	0.451
standard q	-	0.343	0.457	0.521
1	0	0.334	0.459	0.526
1	1	0.337	0.448	0.513
1	2	0.337	0.449	0.512
2	0	0.323	0.456	0.524
2	1	0.334	0.462	0.528
2	2	0.317	0.445	0.513
3	0	0.271	0.414	0.487
3	1	0.306	0.431	0.498
3	2	0.283	0.411	0.479
4	0	0.320	0.447	0.515
4	1	0.314	0.445	0.515
4	2	0.306	0.441	0.510
5	0	0.343	0.462	0.529
5	1	0.326	0.445	0.512
5	2	0.317	0.444	0.515
6	0	0.349	0.463	0.527
6	1	0.334	0.460	0.525
6	2	0.323	0.456	0.523

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Prof. Jacopo Pazzini, for his interest, dedication, and willingness to dive into a topic a bit outside of his usual field. A big thank you also goes to my co-supervisor and company tutor, Stefano Campese, for his constant insights, ideas, and guidance throughout this journey. I'm also grateful to my colleagues at Omnys S.r.l. for making me feel part of the team from day one and to the company for providing the resources and tools that were essential for my thesis work.

A special thanks to my lab mates during my master's, Davide, Guglielmo, and Alessio. We spent two years together working on projects, meeting deadlines, facing exams, and debugging endless lines of code, but most importantly, we built a lasting friendship and shared memories I will always cherish.

Lastly, I want to thank the people back home—my family and friends—who have always supported my decisions and made me proud of the person I've become today.

Thank you all for being part of this chapter in my life.