

RAPPORT DE STAGE

TECHNICIEN

Option : SIC

Elaboré par : Skander Tlili

Stage effectué au sein de Business & AI

du 20/06/2024 au 31/08/2024

Année universitaire : 2024-2025

Réf. : STG_TECH

Dedications

To all those who strive to bridge the worlds of business and artificial intelligence, whose passion for innovation and impact inspire progress each day. To my mentors, who guided me with wisdom and patience, and to my family, for their unwavering support and encouragement.

This work is a testament to the boundless possibilities when dedication meets purpose.

Thanks

I extend my heartfelt thanks to my host organization, Business & AI, for providing me with an environment rich in learning and growth. Special thanks to my colleagues for their insights and support, and to my mentors, who encouraged me at every turn.

Finally, I am grateful to my friends and family, whose faith in me has been my greatest source of strength.

Summary

Dedications	2
Thanks	3
Sommaire	4
Liste des figures	7
Liste des Tableaux	8
Liste d'abréviations	9
INTRODUCTION GENERALE	10
Chapter 1 : Internship Presentation	11
Introduction	11
1. Presentation of the host organization	11
1.1 Who is Business & AI ?	11
1.2 Mission	11
1.3 Partners	12
1.4 Solutions	12
2.My Mission	15
Conclusion	15
Chapter 2 : Prompt Engineering	16
Introduction	16
1. Large Language Models	16
1.1 Natural Language Processing	16
1.1.1 Concept	16
1.1.2 Challenges of NLP	16
1.1.3 Use Cases	16
1.2 Transformers	17
1.2.1 History	17
1.2.2 How do Transformers work?	18
1.2.3 Transformers are language models	18
1.2.4 Transfer Learning	20
1.2.5 General Architecture	21
1.2.6 Attention Layers	22
1.2.7 The original architecture	23
1.2.8 Architectures vs. checkpoints	24
2. Prompt Engineering	25
2.1 Concepts	25
2.1.1 Definition of prompt	25
2.1.2 Definition of Prompt tuning	25
2.1.3 Definition of Prompt Engineering	25
2.2 Reasoning Techniques	26
2.2.1 Zero-Shot Prompting	26



2.2.2 Few shot prompting	27
2.2.3 Chain-of-Thought (CoT) Prompting	27
2.2.4 Self-Consistency Prompting	27
2.2.5 Tree of thoughts	28
2.3 Prompt tuning techniques	29
2.3.1 Active-Prompt	29
2.3.2 Automatic Prompt Engineer	29
2.4 Prompt Engineering techniques	29
2.4.1 Prompt Subtasking	29
2.4.2 Automatic Reasoning and Tool-Use (ART)	31
2.5 Prompt Engineering Frameworks	32
2.5.1 RAG	33
2.5.2 LLM Agents	37
Conclusion	39
Chapter 3 : Solution Implementation	40
Introduction	40
1. Solution Overview	40
1.1 Problem statement	40
1.2 How the App Works :User Perspective	40
1.3 Challenges	41
2. Tools and Dependencies	41
2.1 Development Environment	41
2.2 Programming Language	41
2.3 Libraries and Frameworks	41
3. Solution Architecture	42
3.1 Data Preparation	43
3.1.1 Data Ingestion	43
3.1.2 Data Cleaning	43
3.1.3 Chunking	44
3.2 Sentence Transformation	48
3.2.1 Embedding Model	48
3.2.2 Choosing the embedding model	49
3.2.3 Why I Chosed BERT ?	49
3.2.4 JinaAI as Embedding Model	49
3.2.5 Similarity Calculation	50
3.2.6 Context dependent embeddings vs Context independent embeddings	51
3.3 Index data	52
3.3.1 What is an Index?	53
3.3.2 Indexing	53
3.3.3 Storage	53
3.4 Querying	53
3.4.1 Context Retrieval	54
3.4.2 Response Synthesis	54



3.5 Agent	55
3.5.1 The Need for an Agent	55
3.5.2 Integration with RAG	55
4. Results	54
Conclusion	56
GENARAL CONCLUSION	56
Bibliographic references	57
Netography	58
ANNEXES	59

List of figures

Figure 1 : BUSINESS & AI® Industries and Services areas	11
Figure 2 : Who trusted BUSINESS & AI® (2020)	12
Figure 3 : Organizations using NLP	18
Figure 4 : Evolution of transformers	19
Figure 5 : Illustration of causal Language modeling	20
Figure 6 : Evolution of LM sizes	20
Figure 7 : Pretraining Schema	21
Figure 8 : Transfer Learning Schema	21
Figure 9 : Transformers Architecture	22
Figure 10 : Internal structure of Transformer	25
Figure 11 : Zero shot prompting schema	27
Figure 12 : Chain of thought Schema	28
Figure 13 : Self Consistency Schema	29
Figure 14 : Tree of thoughts schema	29
Figure 15 : The schema of Generated Knowledge prompting	31
Figure 16 : Prompt Chaining schema	31
Figure 17 : Directional Stimulus prompting schema	31
Figure 18 : Schema of Least to Most prompting	32
Figure 19 : RAG workflow	34
Figure 20 : RAG schema	35
Figure 21 : Retriever Schema	35
Figure 22 : Generation Schema	36
Figure 23 : Augmentation Schema	36
Figure 24 : Agent Framework	38
Figure 25 : Data loading	45
Figure 26 : Chunking	45
Figure 27 : text embedding schema	49
Figure 28 : Cosine similarity Formula	51
Figure 29 : Index Schema	54
Figure 30 : Vector Index Schema	55
Figure 31 : Querying schema	56
Figure 32 : Retriever Schema	56
Figure 33 : Response Synthesis schema	57
Figure 34 : Agent with RAG Schema	58

List of Tables

Table 1 : Comparison **53**

Table 2 : Results **59**

List of abbreviations

LLM : Large Language Models

ART : Automatic Reasoning and Tools Use

LM : Language Model

RAG : Retrieval Augmented Generation

NLP : Natural language Processing

GENERAL INTRODUCTION

This report explores the multifaceted journey of an internship centered on the rapidly evolving field of artificial intelligence, specifically focusing on prompt engineering and large language models (LLMs). Beginning with an overview of the host organization, **Business & AI**, the report delves into the company's mission, key partnerships, and innovative AI-driven solutions. The internship's core objective involved gaining hands-on experience in prompt engineering—a crucial technique that enhances the efficiency and reliability of LLMs by structuring inputs for optimal response generation.

In the following chapters, the report systematically covers the fundamental elements of large language models, providing insights into natural language processing (NLP), challenges of the field, and transformative technologies like transformers. This section addresses significant architectural innovations, such as attention layers and transfer learning, that have defined the evolution of NLP systems.

The prompt engineering segment examines foundational concepts, prompt tuning, and reasoning techniques, including zero-shot, few-shot, and chain-of-thought prompting, as well as advanced strategies like tree of thoughts and active-prompt techniques. Additionally, prominent frameworks such as Retrieval-Augmented Generation (RAG) and LLM agents are introduced to illustrate real-world implementations.

The final chapter presents the solution implementation, where data preparation, sentence transformation, and the architecture of the developed system are explained. Concluding with results and reflections, this report highlights the practical application of prompt engineering in developing robust AI solutions and contributes insights into the rapidly expanding capabilities of AI-driven applications.

Chapter 1 : Internship Presentation

Introduction

This chapter provides an overview of my internship experience, focusing on the host organization, Business & AI. It outlines the company's mission, key partners, and the solutions they offer in the AI domain, followed by a detailed description of my mission and assigned tasks.

1. Presentation of the host organization

This provides an overview of Business & AI, starting with a presentation and covers the company's mission, key partners, and the solutions it offers.

1.1 Who is Business & AI ?

BUSINESS & AI is a privately owned company specializing in Artificial Intelligence. With ongoing AI projects in partnership with several financial institutions and leading companies in the European automotive industry. Their AI-powered decision-making solutions enhance efficiency and competitiveness while providing concrete examples of how AI can augment human intelligence. They deliver cutting-edge AI technologies that enable machines to simulate, augment, and at times, surpass human intelligence.

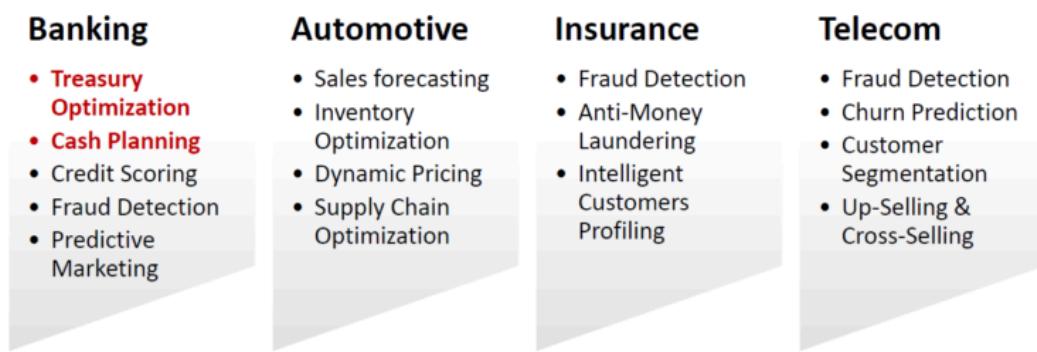


Figure 1 : BUSINESS & AI® Industries and Services areas

1.2 Mission

Their mission is to create a robust connection between the enterprise and the latest scientific breakthroughs in AI. By combining AI, technology, and business expertise, we help companies tackle their most complex business challenges in a cost-effective and non-intrusive manner.

Building a robust bridge between the latest scientific breakthroughs from the field of Artificial Intelligence and the Enterprise.

- Identifying and solving the most complex business challenges faced by the enterprise by combining Artificial Intelligence (AI), Technology and Business expertise.

- Providing cost effective and non-intrusive AI-powered decision-making solutions to help companies get more efficient and more competitive.
- Delivering concrete examples of how AI can make the enterprise smarter by augmenting human intelligence with AI-powered solutions.
- Providing cutting-edge Big Data and AI technologies which enables machines to simulate, augment and sometimes outperform human intelligence.

1.3 Partners

BUSINESS & AI built strong strategic partnerships with both scientific and business partners based in Germany, France, Tunisia, UAE, and Canada.

They are trusted by many large companies ranging from:



Figure 2 : Who trusted BUSINESS & AI® (2020)

BUSINESS & AI built strong strategic partnerships with both scientific and business partners based in Germany, France, Tunisia, UAE, and Canada

1.4 Solutions

1.4.1 Context'AML

The Most Accurate AI-based Behavioral Monitoring Solution for AML/CFT

Context'AML is an AI-powered tool that profiles the financial activities of bank clients, automatically detecting and analyzing Money Laundering and Terrorism Financing risks without predefined scenarios.

Why Context'AML?

Compliance experts face these challenges:

- Valuable resources are wasted on false alerts.
- Real threats are missed due to difficulty filtering true risks.
- Existing AI solutions overlook critical Money Laundering patterns defined by law.

Context'AML Main Benefits:

- Improves Money Laundering risk detection by reducing false alerts.
- Minimizes human involvement with prioritized alert selection.
- Detects hidden patterns by incorporating laws, regulations, and relationships.

The Core of Context'AML :

Context'AML® uses proprietary AI technology combining Deep Active Learning, Unsupervised Learning, and Semi-Supervised Learning. It segments transactions, accounts, and customers into relevant contexts, using Dynamic Temporal Graphs to continuously learn and detect weak signals. This enhances alert relevance compared to rule-based systems.

Context'AML identifies cases needing human investigation, enabling iterative learning, insight capture, and better compliance with AML/CFT regulations.

1.4.2 ATM CASHVISION

Why ATM CASHVISION ?

ATM CASHVISION is a platform that leverages artificial intelligence to revolutionize ATM cash optimization.

Combining powerful Multi-Objective optimization and reinforcement learning algorithms, the platform automatically builds cognitive agents which are capable of: learning, predicting, reasoning and deciding on the best cash replenishment policies to the point of surpassing human operator in many business, operational and financial KPIs.

How does it work ?

What makes ATM CASHVISION Forecasting & Optimization engines powerful is the

innovative combination of many cutting-edge AI techniques". AI & Engineering Teams

Solution benefits

- Reduction of more than 40% of cash requirements
- Elimination of idle cash & opportunity costs
- Reduction of cash transport costs
- 60% reduction in the number of ATM charges
- Less operational risks (counting error, theft, etc.)
- Cash availability rate of more than 98%

1.4.3 CarAdsBot

AI-driven Platform for Competitive Intelligence and Market Trends Analysis of Used Car Ads
Business Problem

Used cars are the second-hand vehicles that are available for resale.



The used car market across the world is growing rapidly. It was valued at \$828.24 billion in 2019, and is projected to reach \$1,355.15 billion by 2027, to register a CAGR of 8.3% from 2020 to 2027.

Today, the growing power of online infrastructure and digitalization, is more and more expanding the used cars market all over the world.

Used Cars buyers, sellers and dealers are often confronted with the challenge of making the right decision from the multitude of car ads and opportunities coming from the different online platforms and websites.

CarAdsBot

CarAdsBot® is an AI-powered platform for competitive intelligence and market trends analysis of used car ads. It continuously and automatically extracts a significantly huge number of ads from all geographical regions. Based on them, it captures and attains enough knowledge and intelligence to guide people to take the optimal and informed decisions upon buying or selling their cars.

Features

- Real-Time Advanced Search
- Real-Time Statistics
- Graphs & Dashboards
- Opportunity Scoring
- Similarity Scoring
- Intelligent Cars Pricing
- Real-Time Price Tracking
- High-upside opportunity Spotting

1.4.4 DEXTIM

Extraction data from images is a challenging problem nowadays because every company wants to digitalize their data. So, we need a tool that can help us to extract information from many types of documents without use of a lot of data.

Due to the lack of data and their variation (types, orientation, bad quality), we need a tool that can perform well from single image and that can deal with the variety of scanned data. Also, the solution must be fast and small sized to give the client the best user experience. Finally, we can't handle unstructured documents without passing first through structured documents that's why our current work will focus on the idea of data extraction from structured documents.

2. My Mission

Throughout my journey at B&AI, I have gained comprehensive knowledge and hands-on experience across several key areas in the domain of artificial intelligence, particularly with large language models (LLMs):

- Understanding Large Language Models (LLMs) and Their Operation : I have developed a deep understanding of how LLMs function, including their internal architecture, training methodologies, and their ability to process and generate human-like text. This involved exploring various transformer-based models, understanding tokenization, attention mechanisms, and the unique challenges of fine-tuning models for specialized tasks.
- Mastering Prompt Engineering Techniques : I dedicated significant time to studying and testing the effectiveness of prompt engineering techniques on smaller-scale LLMs (with fewer than 8 billion parameters). This included experimenting with various prompt structures, exploring ways to elicit more accurate and relevant responses, and refining inputs to enhance model performance on domain-specific tasks.
- In-Depth Exploration of Retrieval-Augmented Generation (RAG) and Intelligent Agents : I dived deep into studying Retrieval-Augmented Generation (RAG), which combines LLMs with external knowledge retrieval mechanisms to improve accuracy and factuality. Additionally, I explored the development and use of intelligent agents to facilitate more interactive and dynamic problem-solving capabilities, including multi-step reasoning and decision-making processes.
- Design and Development of a Q&A Application for PDF Document Processing: I successfully conceived, designed, and developed a question-answering application capable of extracting and processing information directly from PDF documents. Leveraging techniques such as RAG and LLMs, I built a system that enables efficient retrieval of specific answers from large documents, streamlining the process of information discovery and enhancing user experience.

Conclusion

By highlighting both the organization and my mission, this chapter sets the foundation for understanding the scope and impact of the internship. The content prepares the reader for further insights into the practical aspects of my contributions throughout the internship.

Chapter 2 : Prompt Engineering

Introduction

This Chapter delves into the foundational concepts of prompt engineering, with a focus on large language models and their underlying architecture. It covers essential topics in natural language processing (NLP) and explores key reasoning and tuning techniques that enhance the effectiveness of prompt-based interactions.

1. Large Language Models

It is customary to begin the study of any method or technique by first understanding its areas of application and the underlying framework in which it operates. Before exploring effective techniques for prompting large language models, it's crucial to have a foundational understanding of how these systems function.

1.1 Natural Language Processing

Let's start with a brief overview of natural language processing (NLP) which is the field of application of my internship to effectively set the stage for clear understanding of the mission and kind of tasks tackled.

1.1.1 Concept

NLP is a field of linguistics and machine learning focused on understanding everything related to human language. The aim of NLP tasks is not only to understand single words individually, but to be able to understand the context of those words.

1.1.2 Challenges of NLP

Computers don't process information in the same way as humans. For example, when we read the sentence "I am hungry," we can easily understand its meaning. Similarly, given two sentences such as "I am hungry" and "I am sad," we're able to easily determine how similar they are. For machine learning (ML) models, such tasks are more difficult. The text needs to be processed in a way that enables the model to learn from it. And because language is complex, we need to think carefully about how this processing must be done. There has been a lot of research done on how to represent text, and we will look at some methods in the next chapter.

1.1.3 Use Cases :

Transformer models are used to solve all kinds of NLP tasks, like the ones mentioned in the previous section. Here are some of the companies and organizations using Hugging Face and Transformer models, who also contribute back to the community by sharing their models:

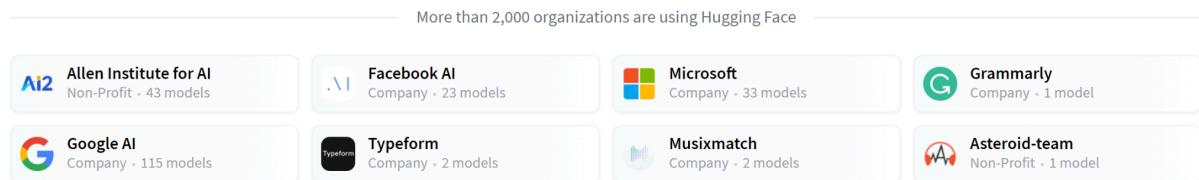


Figure 3 : Organizations using NLP

Before diving into how Transformer models work under the hood, let's look at a few examples of how they can be used to solve some interesting NLP problems.

A. Zero-shot classification

We'll start by tackling a more challenging task where we need to classify texts that haven't been labeled. This is a common scenario in real-world projects because annotating text is usually time-consuming and requires domain expertise. (See [Annex 1](#))

B. Text generation

Now let's see how to use a pipeline to generate some text. The main idea here is that you provide a prompt and the model will auto-complete it by generating the remaining text. This is similar to the predictive text feature that is found on many phones. Text generation involves randomness, so it's normal if you don't get the same results as shown below.

C. Named entity recognition

Named entity recognition (NER) is a task where the model has to find which parts of the input text correspond to entities such as persons, locations, or organizations. Take a look at the example [Annex 3](#)

D. Translation

Translation is a task where the model has to translate a sentence written in a certain human language to another human language, in [Annex 4](#) we'll try translating from French to English.

1.2 Transformers

Transformers are foundational structures of LLMs, they are Deep learning models used to solve all kinds of NLP tasks

1.2.1 History

Here are some reference points in the (short) history of Transformer models:

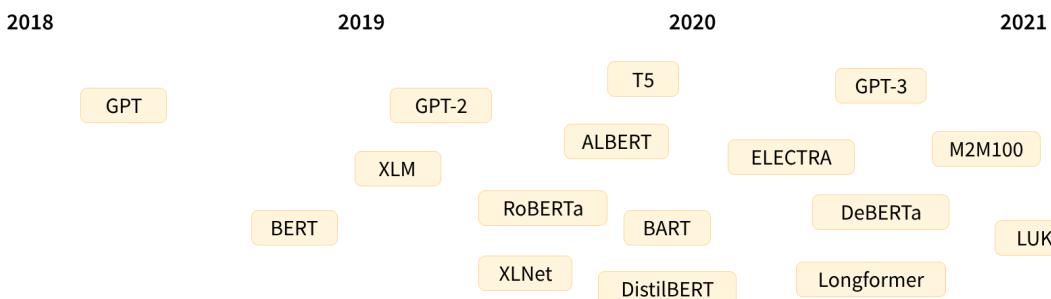


Figure 4 : Evolution of transformers

The Transformer architecture was introduced in June 2017. The focus of the original research was on translation tasks. This was followed by the introduction of several influential models, including:

- June 2018: GPT, the first pretrained Transformer model, used for fine-tuning on various NLP tasks and obtained state-of-the-art results
- October 2018: BERT, another large pretrained model, this one designed to produce better summaries of sentences (more on this in the next chapter!)
- February 2019: GPT-2, an improved (and bigger) version of GPT that was not immediately publicly released due to ethical concerns
- October 2019: DistilBERT, a distilled version of BERT that is 60% faster, 40% lighter in memory, and still retains 97% of BERT's performance
- October 2019: BART and T5, two large pretrained models using the same architecture as the original Transformer model (the first to do so)
- May 2020, GPT-3, an even bigger version of GPT-2 that is able to perform well on a variety of tasks without the need for fine-tuning (called zero-shot learning)

This list is far from comprehensive, and is just meant to highlight a few of the different kinds of Transformer models. Broadly, they can be grouped into three categories:

- GPT-like (also called auto-regressive Transformer models)
- BERT-like (also called auto-encoding Transformer models)
- BART/T5-like (also called sequence-to-sequence Transformer models)

We will dive into these families in more depth later on.

1.2.2 How do Transformers work?

In this section, we will take a high-level look at the architecture of Transformer models.

1.2.3 Transformers are language models

All the Transformer models mentioned above (GPT, BERT, BART, T5, etc.) have been trained as language models. This means they have been trained on large amounts of raw text in a self-supervised fashion. Self-supervised learning is a type of training in which the objective is

automatically computed from the inputs of the model. That means that humans are not needed to label the data!

This type of model develops a statistical understanding of the language it has been trained on, but it's not very useful for specific practical tasks. Because of this, the general pretrained model then goes through a process called transfer learning. During this process, the model is fine-tuned in a supervised way — that is, using human-annotated labels — on a given task.

An example of a task is predicting the next word in a sentence having read the n previous words. This is called causal language modeling because the output depends on the past and present inputs, but not the future ones.

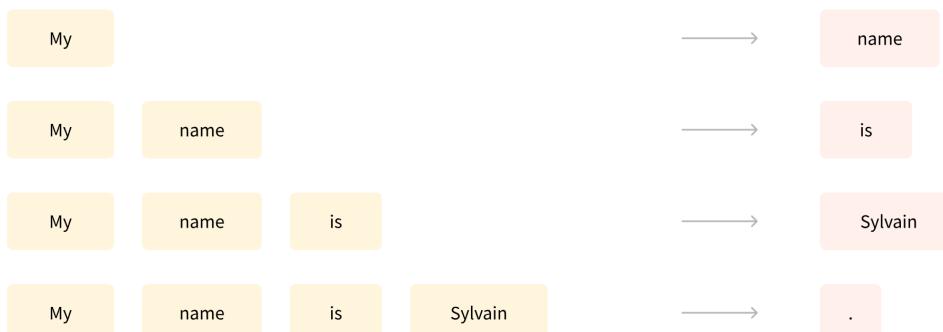


Figure 5 : Illustration of causal Language modeling

Apart from a few outliers, the general strategy to achieve better performance is by increasing the models' sizes as well as the amount of data they are pretrained on. Unfortunately, training a model, especially a large one, requires a large amount of data. This becomes very costly in terms of time and compute resources

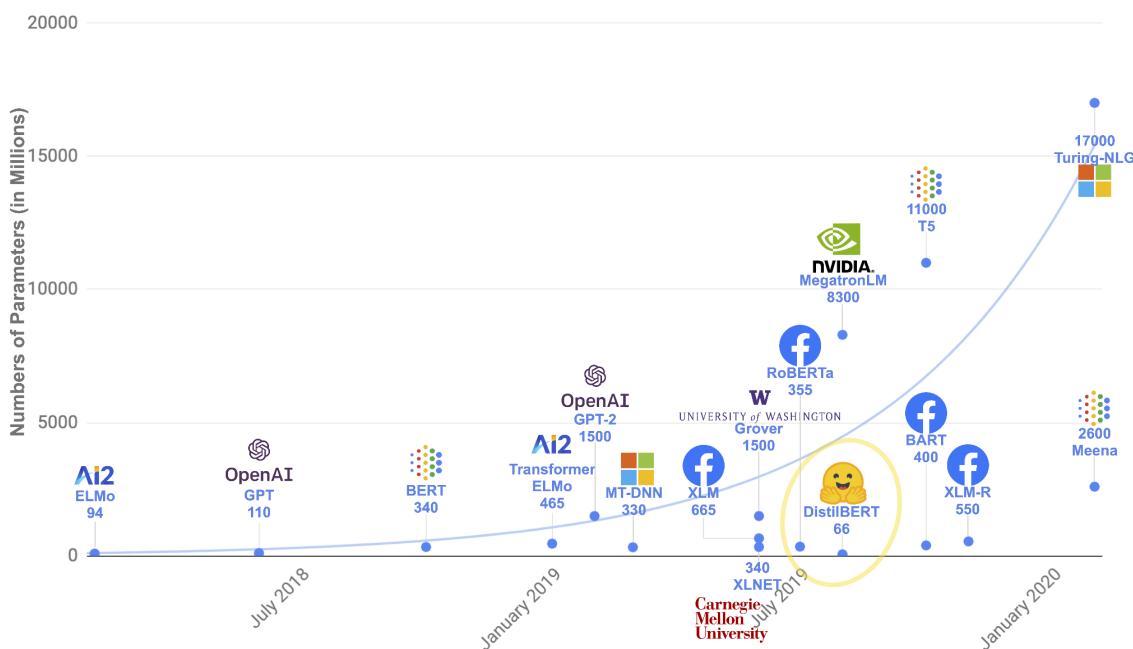


Figure 6 : Evolution of LM sizes

1.2.4 Transfer Learning

Pretraining is the act of training a model from scratch: the weights are randomly initialized, and the training starts without any prior knowledge.

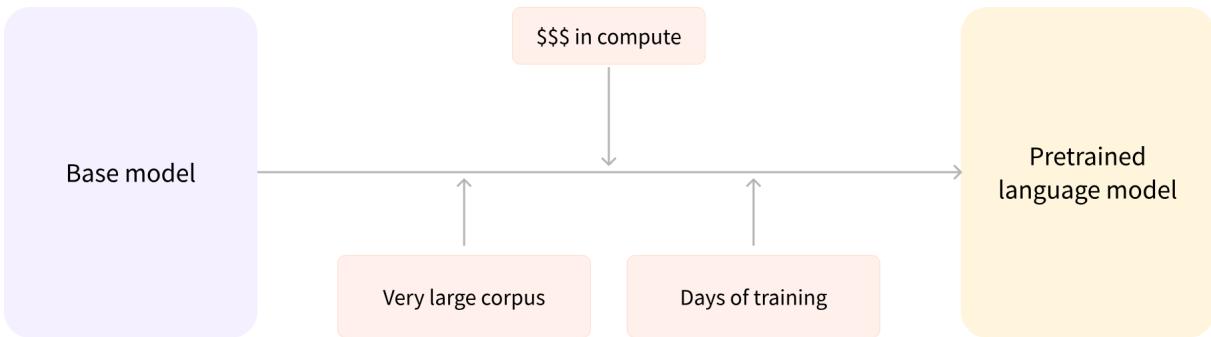


Figure 7 : Pretraining Schema

This pretraining is usually done on very large amounts of data. Therefore, it requires a very large corpus of data, and training can take up to several weeks.

Fine-tuning, on the other hand, is the training done after a model has been pretrained. To perform fine-tuning, you first acquire a pretrained language model, then perform additional training with a dataset specific to your task. Wait — why not simply train the model for your final use case from the start (scratch)? There are a couple of reasons:

The pretrained model was already trained on a dataset that has some similarities with the fine-tuning dataset. The fine-tuning process is thus able to take advantage of knowledge acquired by the initial model during pretraining (for instance, with NLP problems, the pretrained model will have some kind of statistical understanding of the language you are using for your task). Since the pretrained model was already trained on lots of data, the fine-tuning requires way less data to get decent results. For the same reason, the amount of time and resources needed to get good results are much lower. For example, one could leverage a pretrained model trained on the English language and then fine-tune it on an arXiv corpus, resulting in a science/research-based model. The fine-tuning will only require a limited amount of data: the knowledge the pretrained model has acquired is “transferred,” hence the term transfer learning.

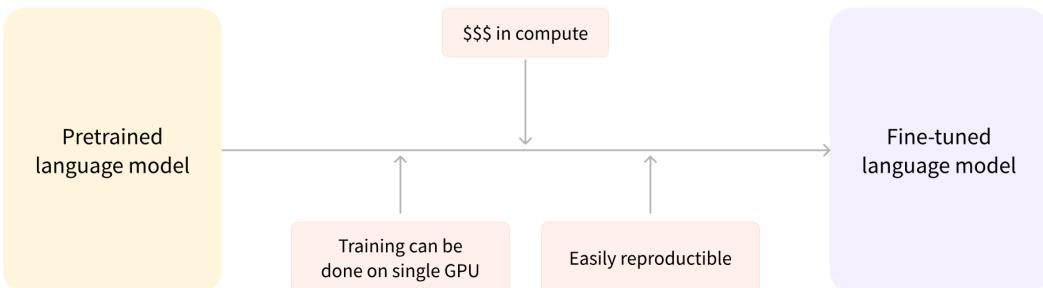


Figure 8 : Transfer Learning Schema

Fine-tuning a model therefore has lower time, data, financial, and environmental costs. It is also quicker and easier to iterate over different fine-tuning schemes, as the training is less constraining than a full pretraining.

This process will also achieve better results than training from scratch (unless you have lots of data), which is why you should always try to leverage a pretrained model — one as close as possible to the task you have at hand — and fine-tune it.

1.2.5 General Architecture

In this section, we'll go over the general architecture of the Transformer model. The model is primarily composed of two blocks:

- Encoder (left): The encoder receives an input and builds a representation of it (its features). This means that the model is optimized to acquire understanding from the input.
- Decoder (right): The decoder uses the encoder's representation (features) along with other inputs to generate a target sequence. This means that the model is optimized for generating outputs.

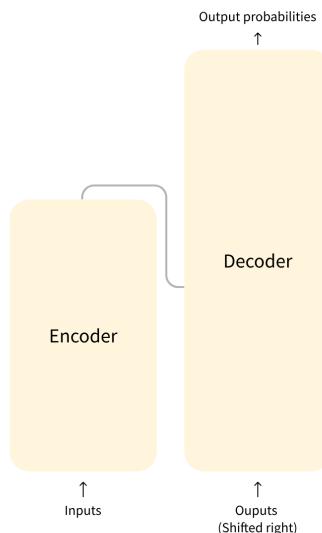


Figure 9 : Transformers Architecture

Each of these parts can be used independently, depending on the task:

- Encoder-only models: Good for tasks that require understanding of the input, such as sentence classification and named entity recognition.
- Decoder-only models: Good for generative tasks such as text generation.
- Encoder-decoder models or sequence-to-sequence models: Good for generative tasks that require an input, such as translation or summarization.

A. Encoder :

Encoder models use only the encoder of a Transformer model. At each stage, the attention layers can access all the words in the initial sentence. These models are often characterized as having “bi-directional” attention, and are often called auto-encoding models.

The pretraining of these models usually revolves around somehow corrupting a given sentence (for instance, by masking random words in it) and tasking the model with finding or reconstructing the initial sentence.

Encoder models are best suited for tasks requiring an understanding of the full sentence, such as sentence classification, named entity recognition (and more generally word classification), and extractive question answering.

Representatives of this family of models include: ALBERT, BERT, DistilBERT, ELECTRA, RoBERTa

B. Decoder :

Decoder models use only the decoder of a Transformer model. At each stage, for a given word the attention layers can only access the words positioned before it in the sentence. These models are often called auto-regressive models. The pretraining of decoder models usually revolves around predicting the next word in the sentence.

These models are best suited for tasks involving text generation. Representatives of this family of models include: CTRL, GPT, GPT-2, Transformer XL

C. Encoder-Decoder :

Encoder-decoder models (also called sequence-to-sequence models) use both parts of the Transformer architecture. At each stage, the attention layers of the encoder can access all the words in the initial sentence, whereas the attention layers of the decoder can only access the words positioned before a given word in the input. The pretraining of these models can be done using the objectives of encoder or decoder models, but usually involves something a bit more complex. For instance, T5 is pretrained by replacing random spans of text (that can contain several words) with a single mask special word, and the objective is then to predict the text that this mask word replaces. Sequence-to-sequence models are best suited for tasks revolving around generating new sentences depending on a given input, such as summarization, translation, or generative question answering. Representatives of this family of models include: BART, mBART, Marian, T5

1.2.6 Attention Layers

A key feature of Transformer models is that they are built with special layers called attention layers. In fact, the title of the paper introducing the Transformer architecture was “Attention Is All You Need”! We will explore the details of attention layers later in the course; for now, all you need to know is that this layer will tell the model to pay specific attention to certain words in the sentence you passed it (and more or less ignore the others) when dealing with the representation of each word.

To put this into context, consider the task of translating text from English to French. Given the input “You like this course”, a translation model will need to also attend to the adjacent word “You” to get the proper translation for the word “like”, because in French the verb “like” is conjugated differently depending on the subject. The rest of the sentence, however, is not

useful for the translation of that word. In the same vein, when translating “this” the model will also need to pay attention to the word “course”, because “this” translates differently depending on whether the associated noun is masculine or feminine. Again, the other words in the sentence will not matter for the translation of “course”. With more complex sentences (and more complex grammar rules), the model would need to pay special attention to words that might appear farther away in the sentence to properly translate each word.

The same concept applies to any task associated with natural language: a word by itself has a meaning, but that meaning is deeply affected by the context, which can be any other word (or words) before or after the word being studied.

Now that you have an idea of what attention layers are all about, let’s take a closer look at the Transformer architecture.

1.2.7 The original architecture

The Transformer architecture was originally designed for translation. During training, the encoder receives inputs (sentences) in a certain language, while the decoder receives the same sentences in the desired target language. In the encoder, the attention layers can use all the words in a sentence (since, as we just saw, the translation of a given word can be dependent on what is after as well as before it in the sentence). The decoder, however, works sequentially and can only pay attention to the words in the sentence that it has already translated (so, only the words before the word currently being generated). For example, when we have predicted the first three words of the translated target, we give them to the decoder which then uses all the inputs of the encoder to try to predict the fourth word.

To speed things up during training (when the model has access to target sentences), the decoder is fed the whole target, but it is not allowed to use future words (if it had access to the word at position 2 when trying to predict the word at position 2, the problem would not be very hard!). For instance, when trying to predict the fourth word, the attention layer will only have access to the words in positions 1 to 3.

The original Transformer architecture looked like this, with the encoder on the left and the decoder on the right:

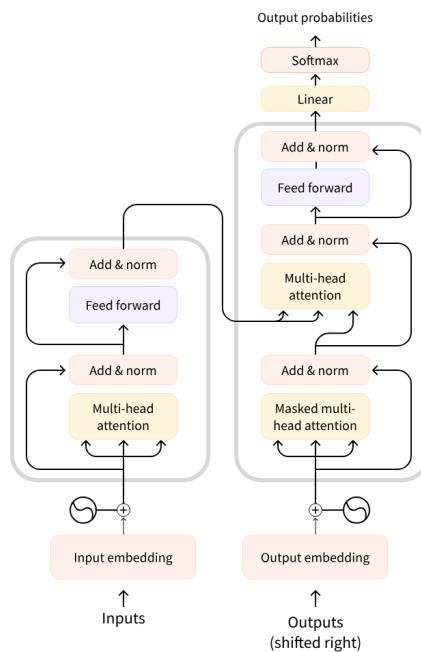


Figure 10 : Internal structure of Transformer

Note that the first attention layer in a decoder block pays attention to all (past) inputs to the decoder, but the second attention layer uses the output of the encoder. It can thus access the whole input sentence to best predict the current word. This is very useful as different languages can have grammatical rules that put the words in different orders, or some context provided later in the sentence may be helpful to determine the best translation of a given word.

The attention mask can also be used in the encoder/decoder to prevent the model from paying attention to some special words — for instance, the special padding word used to make all the inputs the same length when batching together sentences.

1.2.8 Architectures vs. checkpoints

As we dive into Transformer models in this course, you'll see mentions of architectures and checkpoints as well as models. These terms all have slightly different meanings:

- **Architecture:** This is the skeleton of the model — the definition of each layer and each operation that happens within the model.
- **Checkpoints:** These are the weights that will be loaded in a given architecture.
- **Model:** This is an umbrella term that isn't as precise as "architecture" or "checkpoint": it can mean both. This course will specify architecture or checkpoint when it matters to reduce ambiguity.

For example, BERT is an architecture while bert-base-cased, a set of weights trained by the Google team for the first release of BERT, is a checkpoint. However, one can say "the BERT model" and "the bert-base-cased model."

2. Prompt Engineering

In this section I will be covering the theory and the practical aspects of prompt engineering that i've implemented in my solution

2.1 Concepts

Before diving into the theory we need to explain some important concepts

2.1.1 Definition of prompt

A prompt is text written in human language that serves as input to the LLM, it contains information like the instruction or question you are passing to the model and include other details such as context, inputs, or examples.

Prompts are typically composed of several key elements that guide a language model's behavior. These elements include:

1. Instruction – A clear directive that defines the task you want the model to perform. For instance, "Classify the text into neutral, negative, or positive" serves as the instruction in a text classification task.
2. Context – External information that provides additional guidance to the model for generating better responses. Although not present in every prompt, context can significantly improve the model's performance by steering its understanding of the task. For example, adding prior examples of text classification could serve as context to refine the model's outputs.
3. Input Data – This refers to the data you want the model to process. In the text classification example, the input data is "I think the food was okay."
4. Output Indicator – The desired format or type of output. In the same example, the output indicator is "Sentiment:", signaling the model to produce a sentiment-based response.

2.1.2 Definition of Prompt tuning :

It is the art of crafting the best prompt to get the desired output from the LLM. It is a more automated approach that leverages optimization techniques to find the best prompt for a given task. It involves fine-tuning the prompt parameters to maximize the performance of the LLM on the task. Prompt tuning typically requires primarily NLP expertise.

2.1.3 Definition of Prompt Engineering

Prompt Engineering covers all the concepts mentioned above, is the art of crafting and orchestrating an effective series of hierarchical prompts following a certain strategy that guides the LLM towards generating the desired output. It involves understanding the capabilities and limitations of the LLM, identifying the task-specific requirements, and designing prompts that align with both. Prompt engineering requires expertise in NLP, as well as an understanding of the task at hand.

2.2 Reasoning Techniques :

As large language models tackle increasingly complex problems, researchers face the challenge of enhancing their reasoning abilities. Traditional prompting methods often fall short when tasks require multi-step thinking, strategic exploration, or deeper logical connections. To address these limitations, various prompting techniques have been developed to guide models more effectively.

In this discussion, we will explore each of these methods in detail, breaking down their unique approaches to problem-solving.

Full illustrations and examples of these techniques can be found in the annex section.

Not all prompts require every element, and the inclusion of these components depends on the complexity and nature of the task. For instance, a simple prompt may omit context, while more advanced tasks benefit from incorporating all four elements. As seen in the text classification example, prompts can be tailored to the specific use case by adjusting the elements accordingly.

2.2.1 Zero-Shot Prompting

As introduced by [1] Wei et al. (2022), Zero-shot prompting refers to instructing a large language model (LLM) to perform a task without providing any examples or demonstrations. The model relies solely on its training data to interpret the instruction and generate the desired output.



Figure 11 : Zero shot prompting schema

- When to Use: Zero-shot prompting is effective when the task is simple or common enough that the LLM can handle it based on its pre-trained knowledge. For example, in text classification tasks, where the instruction is clear, such as "Classify the text into neutral, negative, or positive," the model can deliver correct results without examples (see [Annex 5](#)).
- When Not to Use: Zero-shot prompting may not work well for more complex or specialized tasks where the model requires additional context or demonstrations to understand the nuances.

2.2.2 Few shot prompting:

Few-shot prompting is a technique that enhances in-context learning by providing demonstrations in the prompt. These demonstrations help condition the model to perform better on specific tasks. By offering a few examples (e.g., 1-shot, 3-shot, or 5-shot), the model can learn how to handle similar inputs and generate the desired output. This method contrasts with zero-shot prompting, where no examples are provided.

Few-shot prompting is particularly useful for tasks that are too complex for zero-shot settings but manageable with minimal guidance. For instance, in language tasks such as using a new word in a sentence, few-shot prompting can steer the model toward better performance. In cases where the model struggles with accuracy, adding examples can help it generalize and improve output consistency.

Few-shot prompting may not be effective for complex reasoning tasks, such as arithmetic or logical reasoning, where examples alone do not sufficiently improve performance. In such cases, advanced techniques like chain-of-thought prompting or fine-tuning may be needed to enhance the model's reasoning abilities. (See [Annex 6](#))

2.2.3 Chain-of-Thought (CoT) Prompting

Chain-of-Thought (CoT) prompting is a method introduced to enhance a language model's reasoning capabilities by encouraging it to break down complex tasks into intermediate reasoning steps. It is particularly useful for tasks that require multi-step reasoning, allowing the model to think through a problem before providing an answer. CoT prompting can be combined with few-shot prompting, where a model is shown a few examples before responding, to improve performance on intricate tasks. (See [Annex 7](#))

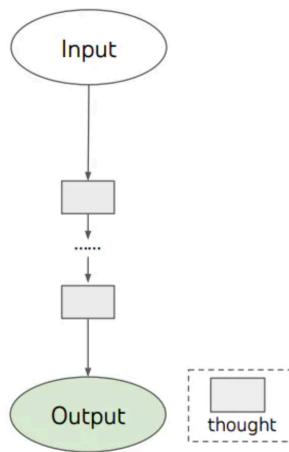


Figure 12 : Chain of thought Schema

2.2.4 Self-Consistency Prompting

Self-consistency is an advanced prompt engineering technique proposed by [2] Wang et al. (2022) to improve the accuracy of Chain-of-Thought (CoT) prompting. Instead of using a

single greedy decoding path, self-consistency generates multiple diverse reasoning paths and selects the most consistent answer among them. This approach boosts performance on tasks involving arithmetic and commonsense reasoning by ensuring the most reliable output is chosen from several plausible options. (See [Annex 8](#))

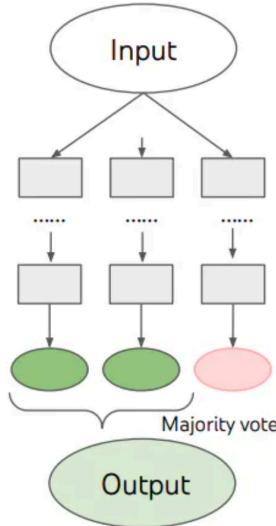


Figure 13 : Self Consistency Schema

2.2.5 Tree of thoughts

The Tree of Thoughts (ToT) framework is an approach to enhance large language models' (LLMs) problem-solving capabilities by structuring the reasoning process as a tree of thoughts. Each "thought" is an intermediate, coherent step toward solving a complex problem, allowing LLMs to explore different solution paths through systematic evaluation and search algorithms like breadth-first search (BFS) or depth-first search (DFS). The framework enables models to self-evaluate their progress, backtrack, and promote viable solutions while eliminating less promising ones. (See [Annex 9](#))

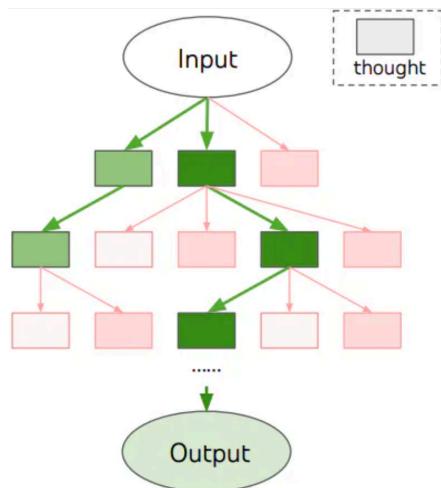


Figure 14 : Tree of thoughts schema

2.3 Prompt tuning techniques :

Here are the most recent techniques to come up with the best candidate prompt

2.3.1 Active-Prompt

Chain-of-thought (CoT) methods rely on a fixed set of human-annotated examples. The problem with this is that the examples might not be the most effective examples for the different tasks. To address this, Diao et al., (2023) recently proposed a new prompting approach called Active-Prompt where The first step is to query the LLM with or without a few CoT examples. k possible answers are generated for a set of training questions. An uncertainty metric is calculated based on the k answers (disagreement used). The most uncertain questions are selected for annotation by humans. The new annotated examples are then used to infer each question. [Annex 10](#) illustrates this approach

2.3.2 Automatic Prompt Engineer

[3] Zhou et al. (2022) propose automatic prompt engineer (APE) a framework for automatic instruction generation and selection. The instruction generation problem is framed as natural language synthesis addressed as a black-box optimization problem using LLMs to generate and search over candidate solutions.

The first step involves a large language model (as an inference model) that is given output demonstrations to generate instruction candidates for a task. These candidate solutions will guide the search procedure. The instructions are executed using a target model, and then the most appropriate instruction is selected based on computed evaluation scores.

[Annex 11](#) illustrates this approach

2.4 Prompt Engineering techniques

In this section i am going to present techniques involving the orchestration of multiple prompts to perform the task

2.4.1 Prompt Subtasking

To enhance the reliability and performance of LLMs, a key technique is breaking down tasks into smaller subtasks in sequential, parallel or hierarchical manner.

This approach makes it easier to identify and fix issues with model responses, allowing for better analysis and improvement at various stages. This technique is especially beneficial for developing LLM-powered conversational assistants, enhancing personalization and overall user experience in applications.

Here are some of the most used examples :

A. Generated Knowledge prompting :

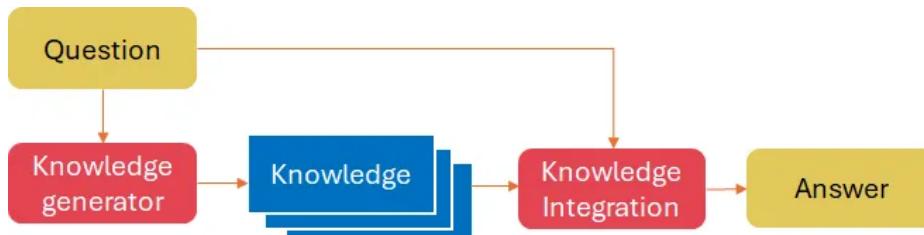


Figure 15 : The schema of Generated Knowledge prompting

This prompting technique under the “Divide and Conquer” concept is Generated Knowledge proposed in [4] Liu et al. (2022). The idea is to use a separate prompt to generate the knowledge first and then use it to get a better response.(See [Annex 12](#))

Generated knowledge prompting involves two stages:

- Knowledge generation: using few-shot demonstrations to generate question related knowledge statements from a language model,
- Knowledge integration: using a second language model to make predictions with each knowledge statement, then selecting the highest-confidence prediction.

B. Prompt Chaining :

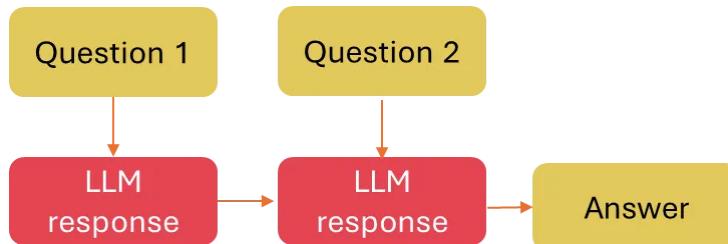


Figure 16 : Prompt Chaining schema

This process, known as prompt chaining, involves prompting the LLM with a subtask and using its response as input for the next prompt. By splitting complex tasks into sequential subtasks, the model can handle them more effectively than with a single, detailed prompt. ([see Annex 13](#))

C. Directional Stimulus Prompting :

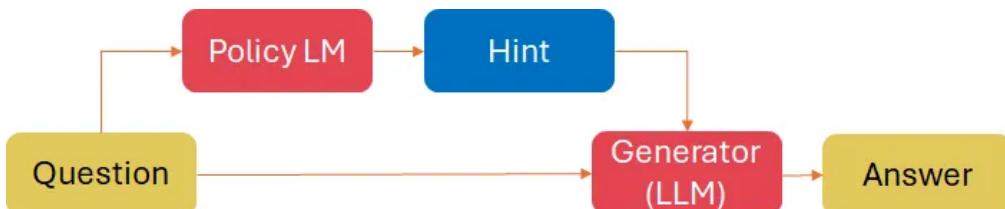


Figure 17: Directional Stimulus prompting schema

Next concept for combining prompts is “Divide and Conquer”. In DSP we have two steps: generate stimulus (e.g., keywords) and use them to improve quality of the response.(See [Annex 14](#))

Directional Stimulus prompting was proposed for summarization, dialogue response generation, and chain-of-thought reasoning tasks. It includes two models:

- a small tunable policy LM is trained to generate a stimulus (a hint),
- a black box frozen large LM is used to generate a summary based on the question and the stimulus from the previous step.

D. Least to most Prompting

Least to Most prompting goes a little bit further adding a step where a model should decide on how to split your task into subproblems.

Experimental results in [4] Zhou et al. (2022) reveal that least-to-most prompting is performing well on tasks related to symbolic manipulation, compositional generalization, and math reasoning. (See [Annex 15](#))

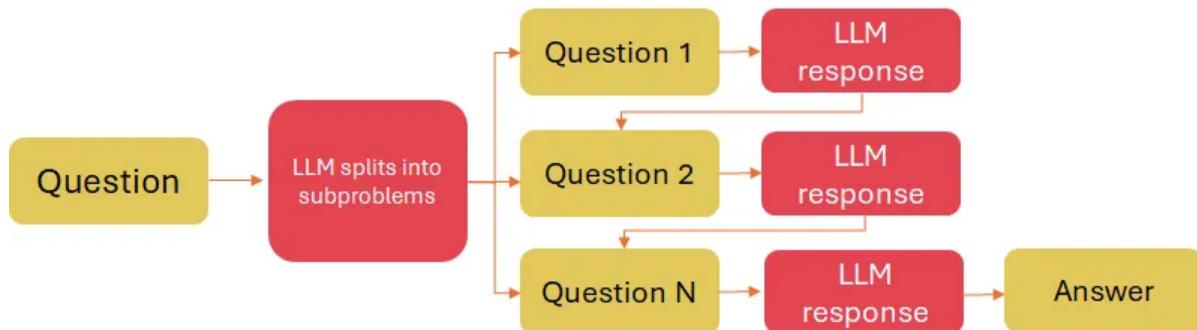


Figure 18 : Schema of Least to Most prompting

2.4.2 Automatic Reasoning and Tool-Use (ART)

ART combines reasoning methods presented above prompting with tool usage in a flexible way, making it effective for various tasks that require external tool usage.

ART is particularly useful when the LLM lacks certain capabilities to perform certain task, or tasks that require extensive computation or tasks that require external knowledge or data.

Examples of tasks include: Mathematical problem-solving, Real time Data, Scientific research queries, Complex decision-making

Tools used in ART include: Calculators, Code interpreters, Search APIs

A. ReAct :

ReAct prompting, introduced by [5] Yao et al. (2022), is a framework where LLMs perform both reasoning (thoughts) and actions in an interwoven manner. This allows the model to not only plan and adapt action steps but also interact with external sources for more accurate information. By generating reasoning traces and actions, ReAct enables LLMs to create action plans, gather information, and update their knowledge dynamically.

The framework improves accuracy, human interpretability, and decision-making by retrieving relevant information while reasoning helps target what to retrieve next. ReAct, especially when combined with chain-of-thought (CoT) prompting, enhances LLMs' ability to use both internal knowledge and external data to deliver more reliable results.

The steps of ReAct reasoning involve alternating between generating thoughts (reasoning) and performing actions. Here is a breakdown of the steps:

1. Prompt with a Task or Question: The model is given a question or task as input.
2. Thought Generation: The model generates a reasoning trace, which includes its plan or approach for solving the task. This could involve breaking down the problem into smaller steps.
3. Action Execution: Based on the generated thought, the model performs an action. This might involve searching for information, interacting with a knowledge base, or retrieving data from an external environment.
4. Observation: The model observes the results of the action (e.g., the information retrieved from a database or search engine).
5. Update Thought: Using the new information from the observation, the model updates its reasoning and adapts its plan.
6. Repeat Thought-Action-Observation Cycle: This process repeats, where the model continues alternating between reasoning and taking actions, until enough information is gathered to solve the problem.
7. Final Answer: The model synthesizes the final solution or answer based on the accumulated information from the reasoning and actions.

This iterative approach ensures that the LLM can reason dynamically and adjust its actions based on external information.(See [Annex 16](#))

B. Reflexion :

Reflexion is a framework that enhances language-based agents through linguistic feedback, allowing them to improve by learning from past mistakes. According to [6] Shinn et al. (2023), Reflexion provides "verbal" reinforcement, where an agent's memory is paired with LLM parameters to guide its decisions. Feedback, either in free-form or scalar form, is transformed into self-reflection, which is used as context for future actions.

The Reflexion framework involves three key models:

1. Actor: Generates actions and text based on observations, creating a trajectory. It uses techniques like Chain-of-Thought (CoT) and ReAct models.
2. Evaluator: Scores the actor's outputs and assigns rewards, using LLMs or rule-based methods.
3. Self-Reflection: Provides verbal feedback to improve the actor's future performance, storing these insights in long-term memory.

The process follows key steps: defining a task, generating a trajectory, evaluating it, reflecting, and then improving the next trajectory. (See [Annex 17](#))

2.5 Prompt Engineering Frameworks

In this section we will go through the theory behind the most frequent prompt engineering frameworks

2.5.1 RAG

In this section I will explain what is RAG, how it works and review the main findings from the survey "Retrieval-Augmented Generation for Large Language Models" [7] Gao et al. (2023), focusing on current methods, advancements, applications, and the technologies behind RAG systems.

What is RAG ?

RAG can be defined as a set of tools that orchestrates prompts by retrieving relevant documents to supplement LLMs with required information for tasks requiring specific knowledge takes

How does it work ?

It takes input and retrieves a set of relevant/supporting documents given a source (e.g., Wikipedia). The documents are concatenated as context with the original input prompt and fed to the text generator which produces the final output. This makes RAG adaptive for situations where facts could evolve over time. This is very useful as LLMs's parametric knowledge is static. RAG allows language models to bypass retraining, enabling access to the latest information for generating reliable outputs via retrieval-based generation.

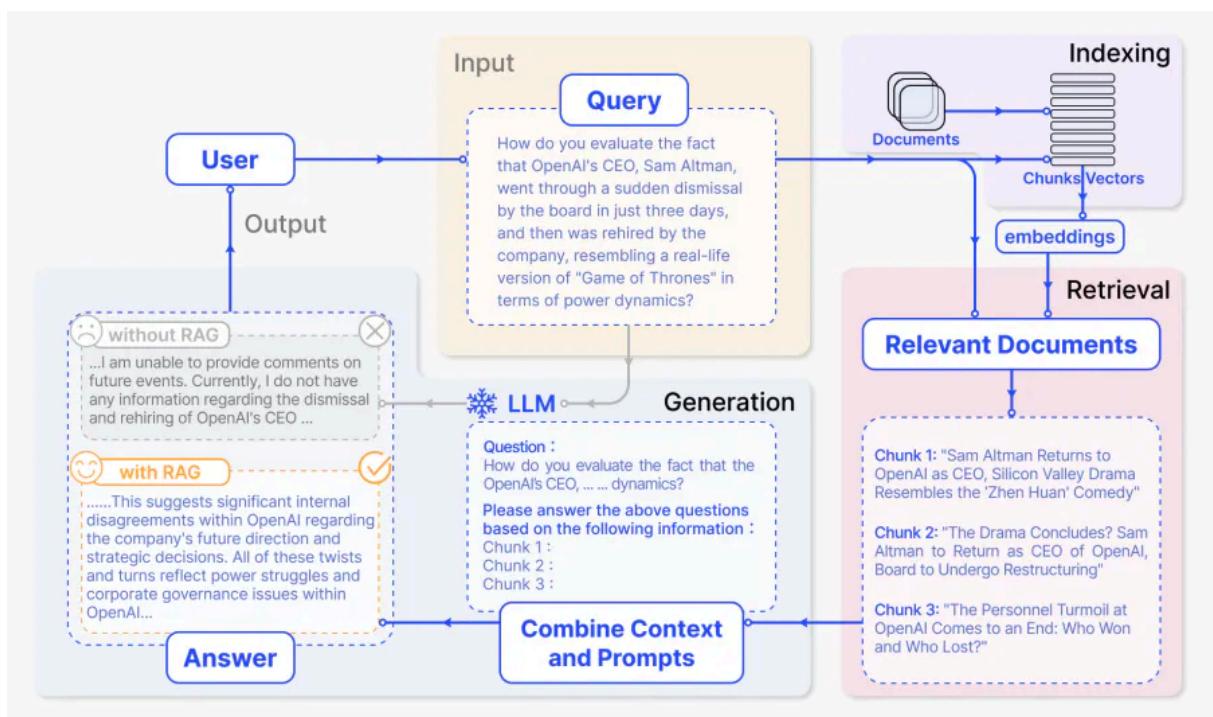


Figure 19 : RAG workflow

RAG operates in four steps:

1. Input: A question or prompt is given to the system.
2. Indexing: Relevant documents are split into chunks, turned into vector embeddings, and stored in a vector database.
3. Retrieval: The system matches the user's query with the indexed documents to find relevant information.

4. Generation: The retrieved documents are combined with the input and passed to the model, which generates a response.

This approach allows language models to provide answers based on the latest information without retraining.

The RAG (Retrieval-Augmented Generation) framework has three main components: retrieval, generation, and augmentation.

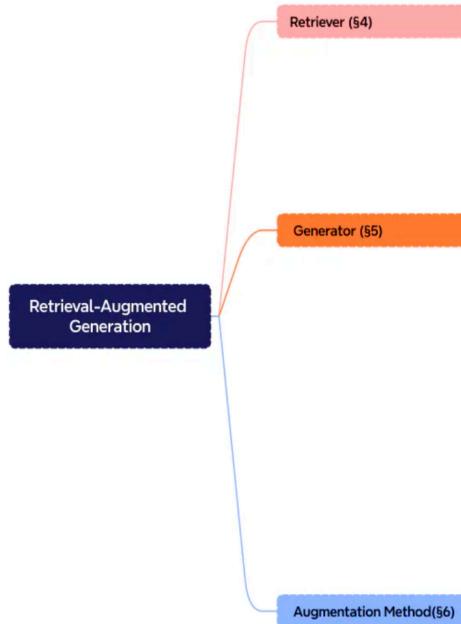


Figure 20 : RAG schema

A. Retrieval

Retrieval is a key part of a Retrieval-Augmented Generation (RAG) system, focused on retrieving relevant context from a retriever. There are three main ways to enhance retrieval:



Figure 21 : Retriever Schema

- Improving Semantic Representations : This involves optimizing how the retriever understands and processes data. Effective chunking is crucial, depending on the model and application. For example, sentence transformers work better with single sentences, while models like text-embedding-ada-002 perform better with larger chunks. Fine-tuning embedding models, especially in specialized domains, can improve accuracy.

- Aligning Queries and Documents : This ensures that user queries match the documents in a semantic space, which may require techniques like query rewriting or embedding transformation to make the queries more precise.
- Aligning Retriever and LLM : This involves aligning retriever outputs with the preferences of large language models (LLMs) using methods like fine-tuning retrievers based on LLM feedback or adding external adapters for better alignment.

B. Generation

The generator converts the retrieved information into coherent text. It can be optimized by:

- Post-retrieval processing : Improving the quality of retrieval results without changing the language model.
- Fine-tuning the LLM : Customizing the model to generate more natural text.

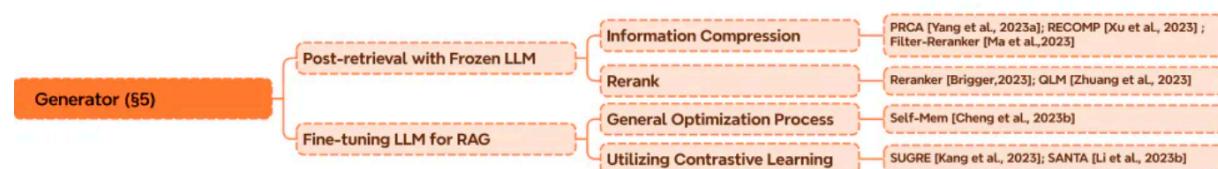


Figure 22 : Generation Schema

C. Augmentation

Augmentation refers to integrating context from retrieved information into a current task. It plays a crucial role in various stages, including pre-training, fine-tuning, and inference.



Figure 23 : Augmentation Schema

- Stages of Augmentation : For instance, RETRO is a system that uses retrieval augmentation during large-scale pre-training by adding an encoder for external knowledge. Fine-tuning enhances Retrieval-Augmented Generation (RAG) systems, while inference techniques help incorporate retrieved data to meet specific task needs.
- Sources of Augmentation: The success of a RAG model depends significantly on the source of augmentation data, which can be unstructured, structured, or generated by large language models (LLMs).
- Augmentation Methods : For complex problems, a single retrieval is often insufficient. Several methods have been proposed:
 - Iterative Retrieval : This method allows multiple retrieval cycles to deepen and enhance relevant information, used in systems like RETRO and GAR-meets-RAG.
 - Recursive Retrieval : This approach uses the output of one retrieval as input for the next, allowing for deeper exploration of information. It is utilized in methods like IRCoT and Tree of Clarifications.
 - Adaptive Retrieval : This method customizes the retrieval process to specific needs, identifying the best times and content to retrieve, as seen in approaches like FLARE and Self-RAG.

RAG Evaluation

Evaluating Retrieval-Augmented Generation (RAG) models is essential for enhancing their performance in various applications. Common metrics for assessment include F1, Exact Match (EM), and frameworks like RaLLe for knowledge-intensive tasks.

RAG evaluation examines both retrieval and generation quality. Retrieval quality is measured using Normalized Discounted Cumulative Gain (NDCG) and Hit Rate, while generation quality considers relevance, harmfulness, and accuracy. Evaluations can be either manual or automatic.

The framework emphasizes three quality scores: context relevance, answer faithfulness, and answer relevance, along with four key abilities: noise robustness, negative rejection, information integration, and counterfactual robustness.

RAG Tools

Some popular comprehensive tools to build RAG systems include LangChain, LlamaIndex, and DSPy. There are also a range of specialized tools that serve different purposes such as Flowise AI that offers a low-code solution for building RAG applications. Other notable technologies include HayStack, Meltano, Cohere Coral, and others. Software and cloud service providers are also including RAG-centric services. For instance, Verba from Weaviate is useful for building personal assistant applications and Amazon's Kendra offers intelligent enterprise search services.

Challenges and Future Directions for RAG:

1. Context Length : As LLMs expand their context windows, RAG systems must adjust to capture the most relevant data.
2. Robustness : RAG needs to better handle misleading or adversarial inputs.
3. Hybrid Approaches : Research focuses on combining RAG with fine-tuned models for improved outcomes.
4. Expanding LLM Roles : Enhancing LLM capabilities is key to advancing RAG systems.
5. Scaling Laws : More exploration is needed on how LLM scaling affects RAG.
6. Production-Ready RAG : Real-world RAG systems must excel in performance, efficiency, security, and privacy.
7. Multimodal RAG : Extending RAG beyond text to domains like images, audio, and video is gaining interest.
8. Evaluation : Better metrics and tools are needed to assess relevance, creativity, and accuracy in complex RAG applications.

2.5.2 LLM Agents

LLM agents are advanced systems that combine LLMs with planning, memory, and tool usage to perform complex tasks. The LLM acts as the main controller, guiding operations to fulfill user requests. For example, if we want to know the average daily calorie intake in the U.S. for 2023, a simple LLM could answer directly if it has the knowledge. However, more complex questions, like analyzing calorie intake trends over a decade and their effects on obesity rates, require a more sophisticated approach.

To tackle such questions, LLM agents can use external resources like health databases and APIs, along with a code interpreter for generating charts. This approach allows the agent to break down tasks into manageable parts and track its progress, ensuring a thorough and accurate response.

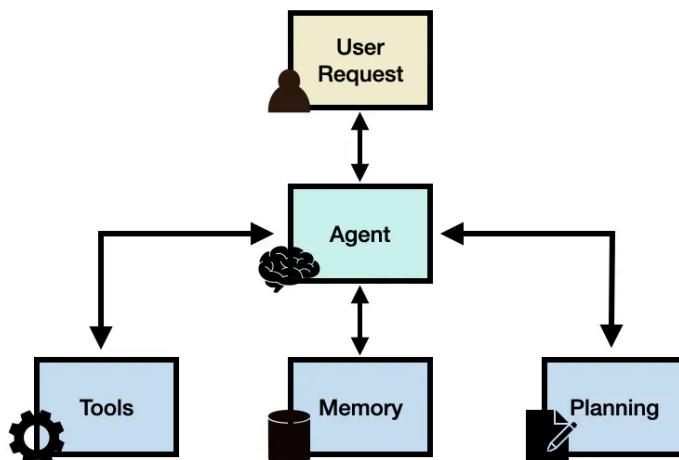


Figure 24 : Agent Framework

LLM Agent Framework

Generally speaking, an LLM agent framework can consist of the following core components:

- Agent/Brain : the agent core acting as coordinator
- Planning : assists the agent in planning future actions
- Memory : manages the agent's past behaviors
- Tools : provide the LLM with new capabilities

A. Agent Brain

An agent in a system is a large language model (LLM) that acts as the main coordinator or brain, guiding operations through prompts that specify its functions and available tools. Although not required, agents can have a defined persona to clarify their role, which is included in the prompt. This profiling may encompass details about the agent's role, personality, and demographics. According to research by Wang et al. (2023), agent profiles can be created through manual design, LLM generation, or data-driven methods.

B. Planning

Planning Without Feedback

The planning or reasoning capability enables the agent to reason better about the problem and reliably find a solution. This module will leverage an LLM to decompose a detailed plan which will include subtasks to help address the user question. Popular reasoning techniques for task decomposition include **Chain of Thought** and **Tree of Thoughts** discussed earlier which can be categorized as single-path reasoning and multi-path reasoning, respectively.

LLM Agent Planning

Planning With Feedback

The planning modules above don't involve any feedback which makes it challenging to achieve long-horizon planning to solve complex tasks. To address this challenge, we can leverage a mechanism that enables the model to iteratively reflect and refine the execution plan based on past actions and observations. The goal is to correct and improve on past mistakes which helps to improve the quality of final results. Two popular methods for this reflection or critical mechanism include ReAct and Reflexion.

C. Memory

Memory helps store an agent's internal logs, including past thoughts, actions, and interactions.

There are two key memory types:

- Short-term memory : Holds information about the agent's current situation, usually via in-context learning, but is limited by the context window.
- Long-term memory : Stores past behaviors and thoughts for long periods, often using external vector stores for quick retrieval.
- hybrid memory : combines both types to improve reasoning and experience accumulation.

Both memory and planning help agents adapt to dynamic environments and recall past actions to plan for the future.

D. Tools

Tools enable LLM agents to interact with external environments like Wikipedia Search API, code interpreters, or math engines. These tools also include databases, knowledge bases, and external models. When using tools, the agent follows workflows to gather information or perform tasks to fulfill user requests. For example, a code interpreter can generate charts based on user queries.

LLMs use tools in various ways:

- MRKL : Combines LLMs with expert modules, such as calculators or APIs.
- Toolformer : Fine-tunes LLMs to use external APIs.
- Function Calling : Adds tool use capability by defining APIs for the model.
- HuggingGPT : Uses LLMs as task planners to connect different AI models to solve tasks.

Conclusion

The chapter concludes by presenting advanced frameworks and tools that optimize prompt engineering, illustrating the growing sophistication and utility of these methods in large language models. These insights are critical for improving NLP applications across various domains.

Chapter 3 : Solution Implementation

Introduction

In this chapter, we will present the implementation phase that will allow the architecture and solution presented in the previous chapters to be realized. We will start with presenting the different tools used, then we will present the architecture chosen for RAG

1. Solution Overview

This project presents an advanced Q&A Agent for Complex Retrieval-Augmented Generation (RAG) Tasks, specifically designed to solve intricate questions that basic semantic retrieval techniques cannot handle. The solution integrates deterministic ReAct Reasoning to orchestrate a controllable autonomous agent, capable of processing complex queries and retrieving answers from custom datasets, particularly PDF documents.

1.1 Problem statement

Traditional retrieval systems often fail when dealing with complex, multi-step questions that require reasoning and interpretation beyond simple keyword matching. This project addresses the challenge of answering non-trivial questions by breaking down queries into smaller, manageable sub-tasks, enabling the agent to provide accurate responses from large datasets.

Here's a user-centric explanation of how the application works

1.2 How the App Works :User Perspective

The application is designed to provide users with an intuitive interface for querying complex information from PDF documents using advanced Retrieval-Augmented Generation (RAG) techniques. Here's a step-by-step overview of the user experience:

A. Setup and Initialization

- Installation: Users start by installing the application on their local machine, following the provided installation steps to set up the environment and dependencies. (See [Annex 18](#))
- Launching the App: After installation, the user runs the application using a simple command in the terminal providing the following arguments :
 - LLM used to power the Agent
 - LLM used for RAG
 - Local Path of PDF document
 - Document Title
 - Description of the PDF document

This launches the Q&A agent interface, ready for interaction. an excerpt of the readme file is provided in [Annex 19](#)

B. Asking Questions

An embedded terminal pops up starting by : “Enter a Question (press q to quit):” where the user can type their query directly into the application interface.

C. Exiting the Application

When users are finished, they can exit the embedded terminal easily by pressing “q”.

This user-centric explanation captures the experience and workflow from the perspective of someone using the application, highlighting its intuitive design and advanced capabilities.

I invite you to take a look at [**Annex 20**](#) for more detail example

1.3 Challenges

Here are the main challenges Addressed by our solution :

- Efficient Chunking : Properly chunking documents to optimize retrieval performance without losing context.
- Threshold-based Retrieval : Implementing a flexible context retrieval strategy based on relevance thresholds instead of simple top-k similarity matches.
- Performance Scaling : Managing agent responses when dealing with large-scale documents and multi-step queries.

2. Tools and Dependencies

The solution leverages several powerful tools and libraries to build an advanced Retrieval-Augmented Generation (RAG) agent capable of handling complex queries. The following tools, frameworks, and development setups were used:

2.1 Development Environment

- Operating System : The application is developed on a Windows machine and can be run on any major operating system (Windows, macOS, or Linux) with Python support.
- Virtual Environment : To manage dependencies, a virtual environment is used, ensuring isolation and consistency across development environments.

2.2 Programming Language

- Python : The core programming language used for this solution is Python. Python’s vast ecosystem of AI libraries, ease of use, and flexibility makes it ideal for building complex agents.

2.3 Libraries and Frameworks

Here are the important libraries used in the development :

- LangChain : Utilized to build the autonomous agent, LangChain simplifies the orchestration of reasoning and action (ReAct) and facilitates interaction between different components.(visit the link in references)
- LlamaIndex : This framework enables efficient retrieval-augmented generation by managing the vector index for document retrieval and query processing.
- Ollama : Used to run local large language models (LLMs), it stands out as a user-friendly, fast, and lightweight tool designed to run large language models (LLMs) locally on your computer. Its versatility is demonstrated by its support for a wide range of LLMs, providing remarkable flexibility to users.

Here are list of the most popular open source LLMs provided by Ollama :

Model	Parameters	Size	Download
Llama 3.2	3B	2.0GB	<code>ollama run llama3.2</code>
Llama 3.2	1B	1.3GB	<code>ollama run llama3.2:1b</code>
Llama 3.1	8B	4.7GB	<code>ollama run llama3.1</code>
Llama 3.1	70B	40GB	<code>ollama run llama3.1:70b</code>
Llama 3.1	405B	231GB	<code>ollama run llama3.1:405b</code>
Phi 3 Mini	3.8B	2.3GB	<code>ollama run phi3</code>
Phi 3 Medium	14B	7.9GB	<code>ollama run phi3:medium</code>
Gemma 2	2B	1.6GB	<code>ollama run gemma2:2b</code>
Gemma 2	9B	5.5GB	<code>ollama run gemma2</code>
Gemma 2	27B	16GB	<code>ollama run gemma2:27b</code>
Mistral	7B	4.1GB	<code>ollama run mistral</code>

Figure : List of the most popular open source models provided by Ollama

- Transformers : Provided by HuggingFace, this library is used for embedding models. The models are crucial for converting text into vector representations, enabling semantic search and retrieval.

3. Solution Architecture

In this section we go through all the steps achieved in my conception development process

3.1 Data Preparation

To build my own Retrieval-Augmented Generation (RAG) application, I'll begin by focusing on data preparation, which includes data ingestion and cleaning to ensure that the RAG model can efficiently locate and generate relevant answers

3.1.1 Data Ingestion :

Every RAG based LLM application starts with smart data choices, which includes :

- **Choose Wisely:** Identify data sources, from portals to APIs, and set up a push mechanism for consistent updates to your LLM app.
- **Governance Matters:** Implement data governance policies upfront. Audit and catalog document sources, redact sensitive data and establish a foundation for context training.
- **Quality Check:** Assess data for diversity, size, and noise levels. Lower-quality datasets dilute responses, necessitating an early classification mechanism.
- **Stay Ahead:** Adhere to data governance even in fast-paced LLM development. It reduces risks and ensures scalable, robust data extraction.
- **Real-Time Cleansing:** Pulling data from platforms like Slack? Filter out noise, typos, and sensitive content in real-time for a clean, effective LLM app.

But in our case this was not quite challenging since because of two main reasons :

One Data type : the solution uses only textual PDF documents

No need for Data Source : The document is provided by the user as Argument
the real challenge lies in the next steps

3.1.2 Data Cleaning

The next step involved transforming every page of my documents into a Document object.

Each document had two key elements: **id, content** and **metadata**.

- The **id** is a unique identifier for each document
- The **page_content** unveils the textual content extracted straight from the document page.
- The **metadata** is a vital ensemble of additional details, like the document's source (the file it originates from), the page number, file type, and other nuggets of information. The metadata keeps tabs on the specific sources it taps into when weaving its magic and generating insightful answers.

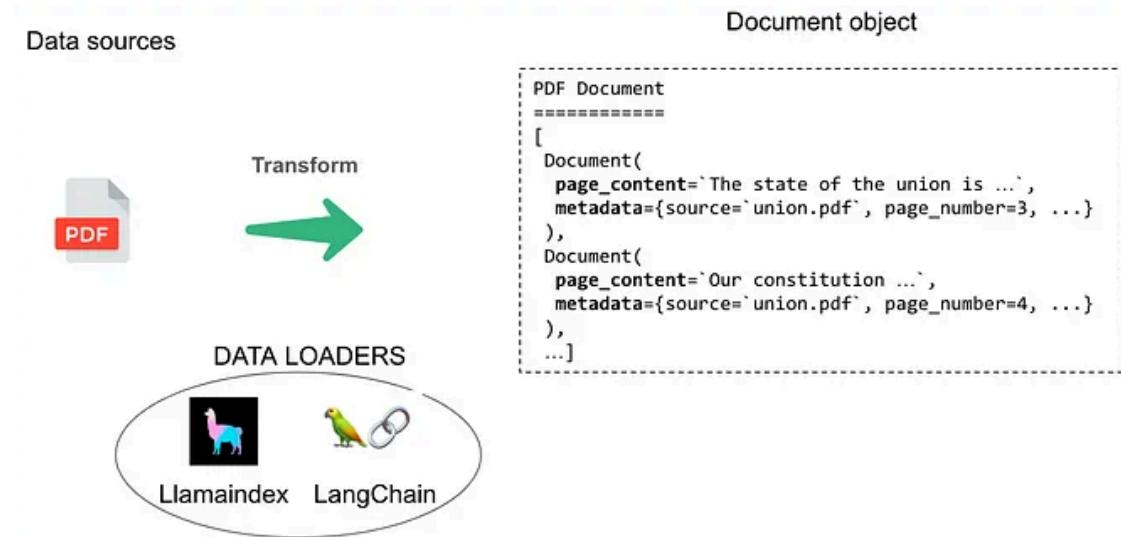


Figure 25 : Data loading

To accomplish this, we leverage robust tools such as Data Loaders, which are offered by open-source libraries like LangChain and Llamaindex. These libraries support various formats, ranging from PDF and CSV to HTML, Markdown, and even databases.

I've provided the piece of code in [Annex 21](#)

3.1.3 Chunking

Chunking is an approach that divides the textual data into smaller, manageable portions to be used as additional context provided to the LLM along the question instead of the entire text corpus of the document

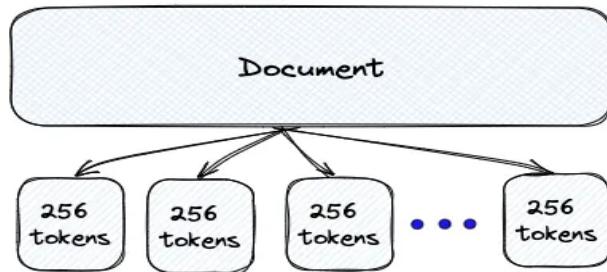


Figure 26 : Chunking

Why Chunking ?

Approaching the task of handling large documents in my Retrieval-Augmented Generation RAG part of the application, I realized that providing an entire document to the model wasn't practical due to the limitations of the model's context window for the following two main reasons:

Here are two key reasons why chunking is essential:

1. Context Window Limitations: LLMs like Llama3.1 or phi3 which I am using have a limited context window, meaning they can only process a certain number of tokens (words) at once, they cannot accept more than 4k tokens in a single prompt. Chunking breaks large documents into smaller, manageable sections that fit within the model's

token limit, allowing it to effectively process and respond to queries based on relevant information.

2. Improved Relevancy and Accuracy: By chunking, you ensure that only the most relevant sections of the document are sent to the model. This reduces the chance of irrelevant information being considered, which helps the model focus on providing accurate and contextually relevant responses to the user's queries.

In later sections I will explain the process of selecting the right chunks of information that matches the question asked

Chunk Size

- **Small chunk size :** For example, a smaller chunk size of 256 has more detailed segments. BUT, this heightened detail brings risk of pivotal info not being included in the most retrieved segments.
- **Large chunk size :** For example, a chunk size of 512 may likely have all vital info within the chunks ensuring a more comprehensive context. BUT it might also slow down the system.

Chunking Methods

In building my RAG application, I explored several chunking methods, each with its strengths and weaknesses, to find the best approach for my use case. After testing different methods, I ultimately implemented **semantic chunking**. Here's a breakdown of the process I went through:

Fixed-size chunking :

We decide the number of tokens in each chunk, throwing in optional overlaps for good measure. Why the overlap? To ensure that the richness of semantic context remains intact across the chunks.

Why go fixed-sized? It's the golden path for most scenarios. Not only is it computationally cheap, saving you processing power, but it's also a breeze to use. No need for complex NLP libraries; just the elegance of fixed-sized chunks seamlessly breaking down your data.

Recursive Character Text Splitting

Recursive Character Text Splitting improves upon basic character splitting by considering the structure of the text. Instead of simply splitting based on a fixed number of characters, this method recursively applies a series of separators (e.g., paragraph breaks, new lines, spaces) to split the text while maintaining its logical structure.

The idea is to begin with larger separators, such as paragraph breaks, and progressively move to smaller ones like spaces or individual characters if necessary. This results in chunks that are more contextually meaningful, such as complete sentences or paragraphs, rather than arbitrary character sequences.

Key Concepts:

- Recursive Splitting: Uses a hierarchy of separators to split text more naturally.

- Separators: Predefined breaking points such as double new lines ("\\n\\n"), single new lines ("\\n"), spaces, and finally individual characters.
- Chunk Size: Maximum number of characters allowed in a chunk.
- Chunk Overlap: Adds redundancy by overlapping portions of consecutive chunks to preserve context.

This approach is particularly useful when chunking for natural language processing tasks that require preserving the meaning of text and ensuring that no meaningful content is cut mid-way, like paragraphs or sentences.(See [Annex 22](#))

Document-Specific Splitting

Document-specific splitting is a technique used to handle various file types and formats beyond standard text documents automatically without more customization. Different document types, such as Markdown files, PDFs, Python code, and others, require unique strategies for splitting content into meaningful chunks. here are few types that i've been testing :

- Markdown Splitting:
Markdown documents use specific delimiters (e.g., headers, code blocks, horizontal lines) to separate content. Chunking strategies can utilize these delimiters to segment content more naturally.(See [Annex 23](#))
- PDF Splitting:
PDFs often contain structured data such as tables. Traditional text-based splitting methods do not work well for such formats. Libraries like Unstructured can be employed to handle complex elements like tables and extract them in formats that are easier for models to parse.

In each case, the goal is to adapt the splitting method to the nature of the document to preserve logical units of information, which is crucial for efficient processing by language models.

In [Annex n](#) i've presents a code demonstrating how different delimiters are used to chunk Markdown. This splitting works by detecting headers and other structural elements

Semantic Chunking

Isn't it weird that we have a global constant for chunk size? Isn't it even weirder that our normal chunking mechanisms don't take into account the actual content? There has to be a better way - let's explore and find out.

Embeddings represent the semantic meaning of a string. They don't do much on their own, but when compared to embeddings of other texts you can start to infer the relationship between chunks. I want to lean into this property and explore using embeddings to find clusters of semantically similar texts.

The hypothesis is that semantically similar chunks should be held together.

I tried a walk method. I started at the first sentence, got the embedding, then compared it to sentence #2, then compared #2 and #3 and so on. I was looking for "break points" where

embedding distance was large. If it was above a threshold, then I considered it the start of a new semantic section. I originally tried taking embeddings of every sentence, but this turned out to be too noisy. So I ended up taking groups of 3 sentences (a window), then got an embedding, then dropped the first sentence, and added the next one. This worked out a bit better.

3.2 Sentence Transformation

In order to effectively manage large documents within my RAG application, chunking serves as a crucial initial step. However, once these chunks are identified, the next challenge is to search for the most relevant sections based on a user's query. To achieve this, we need to convert the textual chunks into a format suitable for comparison and retrieval. This is where vector embeddings come into play. By transforming the text into numerical vectors using embedding models, we can measure the similarity between query and document chunks, ensuring that the system retrieves the most contextually relevant information for downstream tasks like question answering and information retrieval.

3.2.1 Embedding Model

Embeddings are a type of word representation (with numerical vectors) that allows words with similar meanings to have a similar representation.

These vectors can be learned by a variety of machine-learning algorithms and large datasets of texts. One of the main roles of word embeddings is to provide input features for downstream tasks like text classification and information retrieval.

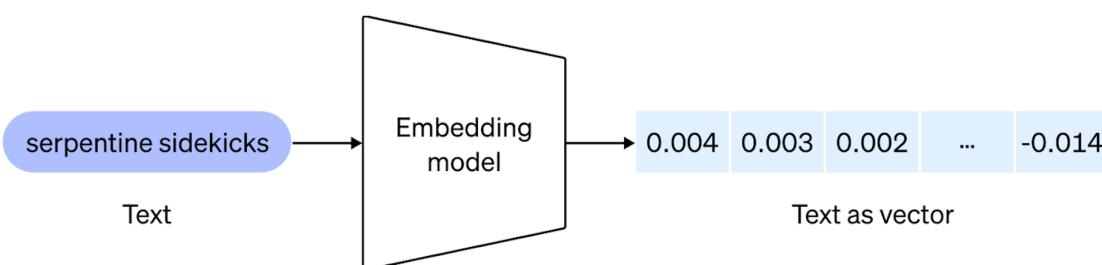


Figure 27 : text embedding schema

3.2.2 Choosing the embedding model

To make an informed choice, I explored two main approaches:

- Using Academic Benchmarks: Academic benchmarks can offer a quick snapshot of how models perform across common tasks. However, I realized that these benchmarks often don't fully capture the nuances and real-world usage of retrieval systems in AI applications. While they provided a good starting point, they weren't enough on their own.
- Model Testing and Evaluation: I took the more hands-on approach by testing multiple embedding models. This allowed me to assess which model best suited my specific use case. Here's an overview of the process I followed and why I selected BERT over the other options.

3.2.3 Why I Chosed BERT ?

After testing these models, I ultimately decided to use BERT based encoder offer by JinaAi and called “jina-embeddings-v2-base-en” for the following several reasons:

- Rich Contextual Understanding: Unlike context-independent embeddings, BERT's bi-directional architecture allows it to capture the context from both directions (left-to-right and right-to-left) in a sentence. This bidirectional nature makes BERT's embeddings far more nuanced, especially when dealing with complex sentences and words that have multiple meanings.
- Strong Performance on Downstream Tasks: BERT's pre-training on the Masked Language Model (MLM) and Next Sentence Prediction (NSP) tasks gives it a deep understanding of word relationships and sentence structures, making it ideal for downstream tasks like question-answering and natural language inference.
- Flexibility in Application: BERT can be fine-tuned for a variety of NLP tasks, making it a versatile model for tasks such as text classification, named entity recognition, and information retrieval, all of which were relevant to my use case.
- Large Pre-training Corpus: BERT is pre-trained on an extensive dataset, including the Books Corpus and English Wikipedia, which provides a solid foundation for understanding a wide range of topics. This broad pre-training made it a powerful tool for handling diverse types of text.
- Powerful Transformer Architecture: BERT's multi-layered transformer architecture with self-attention mechanisms allows it to learn deep representations of text, outperforming LSTM-based models like ELMo and CoVe in terms of both accuracy and efficiency.

3.2.4 JinaAI as Embedding Model

jina-embeddings-v2-base-en is an English, monolingual embedding model supporting 8192 sequence length. It is based on a BERT architecture (JinaBERT) that supports the symmetric bidirectional variant of ALiBi to allow longer sequence length. The backbone

jina-bert-v2-base-en is pretrained on the C4 dataset. The model is further trained on Jina AI's collection of more than 400 millions of sentence pairs and hard negatives. These pairs were obtained from various domains and were carefully selected through a thorough cleaning process.

The embedding model was trained using 512 sequence length, but extrapolates to 8k sequence length (or even longer) thanks to ALiBi. This makes our model useful for a range of use cases, especially when processing long documents is needed, including long document retrieval, semantic textual similarity, text reranking, recommendation, RAG and LLM-based generative search, etc.

With a standard size of 137 million parameters, the model enables fast inference while delivering better performance than our small model. It is recommended to use a single GPU for inference.

3.2.5 Similarity Calculation :

Similarity refers to the mathematical comparison of two vectors in a vector space to determine how alike they are. In my case, I needed a reliable way to evaluate how close a query vector (representing a user's input) is to the stored vectors in my database. This helps retrieve the most relevant information or documents based on their semantic meaning.

To achieve this, I chose Cosine Similarity as my primary similarity measure.

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

Figure 28 : Cosine similarity Formula

Cosine Similarity measures the cosine of the angle between two vectors, ranging from -1 to 1. A value of 1 means the vectors are identical (pointing in the same direction), 0 means they are orthogonal (completely unrelated), and -1 means they are diametrically opposed.

Here's why I chose Cosine Similarity for my application:

- Length Independence: Cosine similarity focuses on the direction of vectors rather than their magnitude. This is crucial in cases where the document or data length varies. For example, in text-based applications, two documents with similar content but different lengths can still have a high cosine similarity.
- Semantic Matching: My goal was to match the meaning and context of the data, not just the absolute values. Since cosine similarity captures the angle between vectors, it's well-suited for understanding the semantic relationships between queries and stored data.

3.2.6 Context dependent embeddings vs Context independent embeddings

In my application, I've tested JinaAi on two distinct embedding methods:

A. Context-Independent Embeddings:

Initially, I utilized context-independent embeddings, where each chunk is assigned a fixed vector, irrespective of the entire context of the document.

B. Context-Dependent Embeddings:

Subsequently, I transitioned to context-dependent embeddings, where the text chunk representation changes based on the entire context of the document.

C. Comparison

In this comparison, we processed a PDF document containing information about Canada by dividing it into multiple chunks. For each chunk, we calculated embeddings using two distinct methods: context-independent and context-dependent embeddings. This resulted in each chunk having two different embeddings based on the method used. Additionally, we calculated the embedding of a specific query related to "Cultural Monuments." To evaluate the effectiveness of each embedding method, we computed the similarity between the query's

embedding vector and the embedding vectors of all the chunks. While the full comparison involves all document chunks, we have chosen to present only chunks 7 and 13 due to the clarity and significance of their results in illustrating the differences between the two embedding approaches.

Query = "Cultural Monuments"

Chunk ID	Chunk text	Context Independent Similarity	Context Dependent Similarity
Chunk 7	"Canada is home to North America's only remaining walled city—the historic neighborhood of Old Quebec, also the first city in North America to be placed on UNESCO's World Heritage Sites list."	0.7346594	0.67476344
	"Ottawa, Canada's capital region, houses the country's most precious cultural treasures."	0.7257095	0.789799

Table 1 : Comparison

D. Observation and Interpretation

The comparison between context-independent and context-dependent embeddings reveals significant insights into their performance. In analyzing the similarity scores for the query "Cultural Monuments," we observe distinct behavior between the two chunking methods.

- Chunk 7 demonstrates a similarity score of 0.7346594 with context-independent embeddings, indicating a strong semantic alignment with the query. However, its context-dependent counterpart scores only 0.67476344. This discrepancy suggests that while context-independent embeddings capture the essence of the content effectively, context-dependent embeddings are overly reliant on specific keyword matching.
- Chunk 13, in contrast, showcases a higher similarity score of 0.789799 with context-dependent embeddings, primarily due to the presence of the keyword "cultural." Although this chunk scores higher, it is essential to note that its content is less semantically aligned with the overall theme of cultural monuments when compared to Chunk 7. This highlights the fundamental limitation of context-dependent embeddings: they can skew results based on the presence of certain words, leading to an "outfitting" effect where the embeddings become overly tailored to the query rather than reflecting true semantic relevance.

E. Conclusion

The analysis demonstrates that while context-dependent embeddings can yield higher similarity scores under specific conditions, they are not always reliable indicators of semantic relevance. The skewed results for Chunk 13 emphasize how the outfitting of

context-dependent embeddings can lead to misleading conclusions based solely on keyword presence. In contrast, context-independent embeddings provide a more balanced and robust assessment of content relevance, making them the preferable choice for capturing the nuanced meanings within the text. Therefore, it is recommended to utilize context-independent embeddings for a more accurate representation of semantic similarity.

3.3 Index data

Now that we have our embedded chunks, we need to index (store) them somewhere so that we can retrieve them quickly for inference.

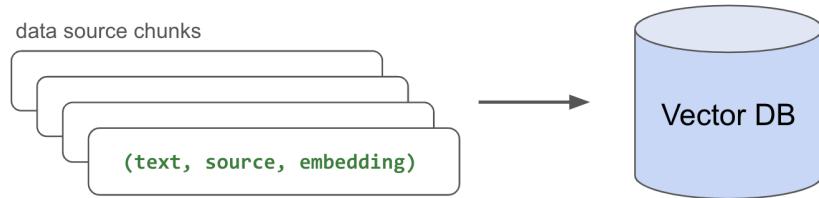


Figure 29 : Index Schema

3.3.1 What is an Index?

An Index is a data structure composed of Document objects, designed to enable querying by an LLM. Our Index is designed to be complementary to your querying strategy.

3.3.2 Indexing

I've used the Vector Store Index method because it stores indexed data in a data structure called nodes. these nodes map each document/chunk to its corresponding to later be stored in a vector DataBase or Persisting it to disc depending on our choice.

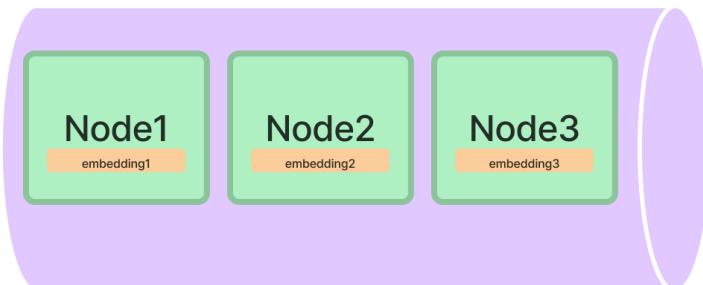


Figure 30 : Vector Index Schema

This allowed me to query by fetching the most similar nodes based on the embeddings, which was essential for achieving semantic search. The ability to directly retrieve the most relevant nodes based on the vector similarity made this method highly efficient for my needs, particularly when dealing with unstructured data and complex queries.

3.3.3 Storage

Since we are not, we are not working with a huge amount of data. We do not need a Database management system for storing our vectors and chunks. Hence, The simplest way to store our indexed data is to use the built-in `.persist()` method of every Index, which writes all the data to disk at the location specified. This works for any type of index.

3.4 Querying

Querying is the process of retrieving the chunks that are most similar to the user question (chunks that contain the information required for response) and feed them to LLM along with the user query or question.

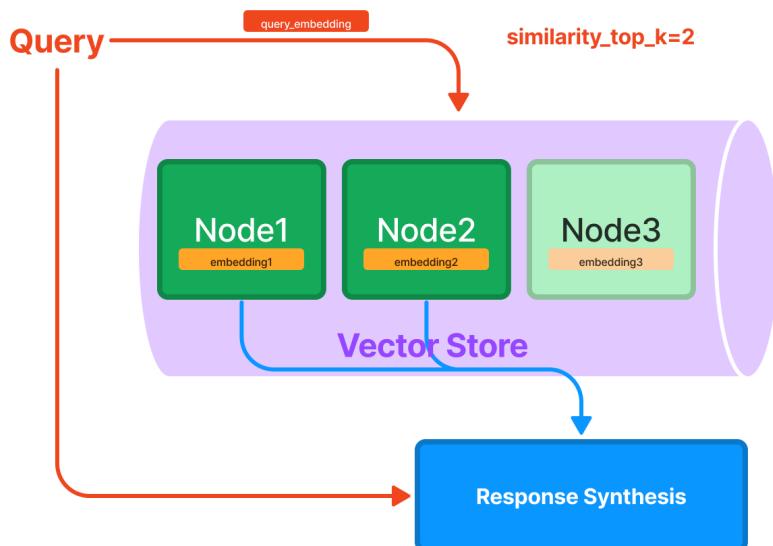


Figure 31 : Querying schema

This process is achieved through two main steps :

3.4.1 Context Retrieval

With our embedded chunks indexed in our vector database, we're ready to perform retrieval for a given query. We'll start by using the same embedding model we used to embed our text chunks to now embed the incoming query

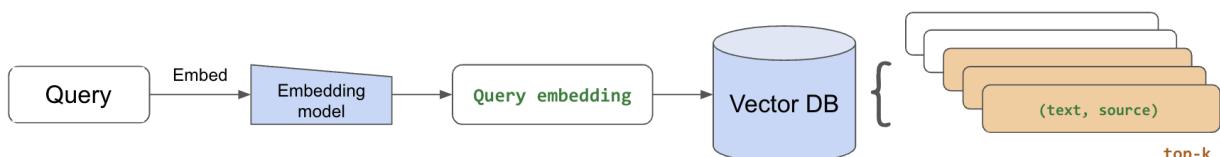


Figure 32 : Retriever Schema

Then, we'll retrieve the most relevant chunks by extracting the most similar embedded chunks based on a threshold to our embedded query. We use cosine distance but there are many options to choose from. Once we retrieve the top num_chunks, we can collect the text for each chunk and use it as context to generate a response.

3.4.2 Response Synthesis

We can now use the context to generate a response from our LLM. Without this relevant context that we retrieved, the LLM may not have been able to accurately answer our question. And as our data grows, we can just as easily embed and index any new data and be able to retrieve it to answer questions.

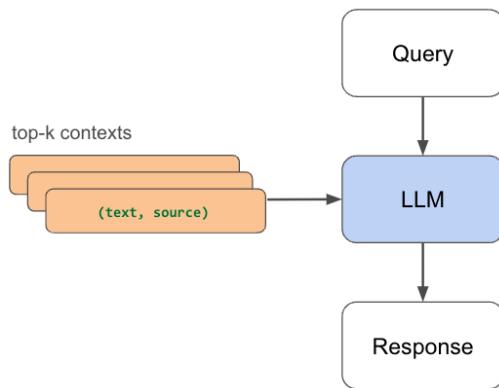


Figure 33 : Response Synthesis schema

3.5 Agent

In addition to building a robust RAG system, I also implemented an agent-based architecture that integrates reasoning and acting capabilities, utilizing the ReAct paradigm. This agent serves as an intelligent orchestrator, leveraging the power of RAG for retrieving relevant information while using advanced decision-making and reasoning processes to answer the questions.

3.5.1 The Need for an Agent

The RAG system excels at retrieving external knowledge to enhance responses by grounding them in factual information. However, certain tasks, especially those requiring a sequence of actions, planning, or interactive feedback from an external environment, require more than simple retrieval. For these tasks, a language model needs to reason about the problem, execute actions in response to dynamic environments, and update its internal understanding based on feedback. This is where the ReAct agent comes into play.

3.5.2 Integration with RAG

The RAG system was embedded as a tool within the ReAct agent's environment. The ReAct agent uses the RAG retriever as an external source for fetching necessary information, helping the agent ground its reasoning in relevant, up-to-date data. For example, in a case where the agent needs additional data about a technical subject:

- The agent identifies the need for more information.
- It sends a request to the RAG system to retrieve relevant documents.
- Once the information is retrieved, the agent incorporates this data into its reasoning process before proceeding with its final task execution.

This integration not only enables more dynamic task-solving but also enriches the reasoning capabilities of the model by allowing it to act upon new information rather than relying solely on pre-trained knowledge.

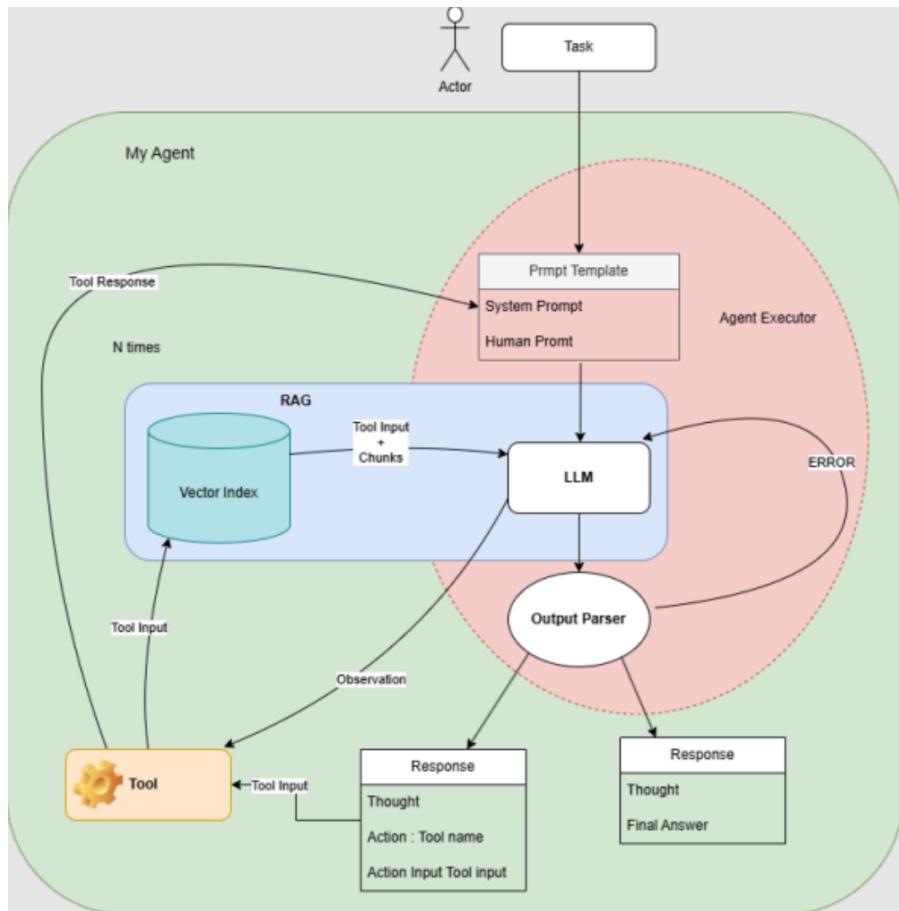


Figure 34 : Agent with RAG Schema

4. Results

The following table presents the results of the Q&A functionality testing for the RAG-based LLM application, using the user guide for Composable AI as the knowledge source. This table is an excerpt of a CSV file consisting of 68 entries across three columns: Question, Answer, and RAG_Answer.

- "Question" : column represents the input queries posed to the system, while
- "Answer" column reflects the expected correct response based on the content of the document.
- "App_Answer" column displays the response generated our application

These results will give insight into the application's accuracy and alignment between expected answers and generated responses.

Question	Answer	App Answer
What is the main difference between Composable AI and Isolated AI?	Composable AI is a modular approach that allows for the integration of different AI models, while Isolated AI is	;Composable AI and Isolated AI differ in their approach to collaboration, flexibility, and integration.

	a single, standalone AI system	Composable AI allows for collaboration between different AI assets, minimizing failure risk by seeking help from underperforming ones and offering assistance when stable. In contrast, Isolated AI refers to systems that work independently
What are the key benefits of using Composable AI?	Composable AI offers several benefits, including faster time to market, increased flexibility, and a quicker payback on investments	Composable AI offers a flexible, modular approach that helps organizations quickly generate value from AI investments while scaling for long-term success. Its ability to adapt to changing market conditions, integrate with legacy systems, and optimize costs makes it more efficient than traditional AI methods, enabling faster implementation and continuous innovation.
What are the main challenges associated with adopting Composable AI?	One of the main challenges is ensuring that the data used by the different AI models is accurate and consistent; Composable AI is designed to bridge the gap between potential of AI and real-world challenges.	The main challenge associated with adopting Composable AI lies in overcoming these real-world obstacles that often defeat traditional approaches. These challenges include poor data quality, weak infrastructure, difficulties integrating AI, and high costs.

Table 2 : Results

Conclusion

In summary, the chapter provides a comprehensive walkthrough of the solution's implementation, covering each critical phase from data processing to response generation. The results section highlights the effectiveness of the applied methods in addressing the problem statement.

GENERAL CONCLUSION

In conclusion, this internship provided me with a comprehensive overview of how cutting-edge AI principles and prompt engineering techniques can drive innovation in NLP applications. Chapter 1 outlined the foundational role of Business & AI, highlighting its mission, key partnerships, and suite of solutions that align well with advancing AI capabilities. Chapter 2 dove deeply into the technical landscape of prompt engineering and large language models, presenting essential concepts, from the nuances of NLP challenges to the architecture of transformers and advanced reasoning techniques like Chain-of-Thought and Self-Consistency Prompting. It demonstrated how these techniques shape the AI models' performance and adaptability, allowing for more precise, context-aware language responses. Chapter 3 brought theoretical knowledge into practice by implementing a solution grounded in AI principles. Through rigorous data preparation, architecture design, and innovative use of embedding models, the solution tackled real-world challenges and offered insights into context-dependent vs. independent embeddings. The project's results confirmed the value of prompt engineering and framework-based architectures, such as RAG and LLM agents, in creating sophisticated, responsive AI applications. This internship has equipped me with valuable practical skills and a strategic understanding of deploying AI solutions, positioning me well for future contributions in the rapidly evolving field of artificial intelligence.

Bibliographic references

- [1] Jason Wei*, Maarten Bosma*, Vincent Y. Zhao*, Kelvin Guu*, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le FINETUNED LANGUAGE MODELS ARE ZERO-SHOT LEARNERS arXiv:2109.01652v5 [cs.CL] 8 Feb 2022
- [2] Wang et al. (2022) Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, Denny Zhou Self-Consistency Improves Chain of Thought Reasoning in Language Models Tue, 7 Mar 2023
- [3] Zhou et al. (2022) Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, Jimmy Ba Large Language Models Are Human-Level Prompt Engineers Thu, 3 Nov 2022
- [4] Liu et al. (2022) Jiacheng Liu, Alisa Liu, Ximing Lu, Sean Welleck, Peter West, Ronan Le Bras, Yejin Choi, Hannaneh Hajishirzi Generated Knowledge Prompting for Commonsense Reasoning, arXiv:2110.08387v3 [cs.CL] 28 Sep 2022
- [5] Yao et al. (2022) Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, Yuan Cao ReAct: Synergizing Reasoning and Acting in Language Models Fri, 10 Mar 2023
- [6] Shinn et al. (2023) Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, Shunyu Yao Reflexion: Language Agents with Verbal Reinforcement Learning arXiv:2303.11366v4 [cs.AI] 10 Oct 2023
- [7] Gao et al. (2023) Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, Haofen Wang Retrieval-Augmented Generation for Large Language Models: A Survey Mon, 18 Dec 2023



Netography

Prompt Engineering Guide | Prompt Engineering Guide (promptingguide.ai) :
<https://www.promptingguide.ai/>

Introduction - Hugging Face NLP Course : <https://huggingface.co/learn/nlp-course/>

Building an LLM Application - LlamaIndex :
<https://docs.llamaindex.ai/en/stable/understanding/>

Prompt Engineering: Classification of Techniques and Prompt Tuning | by Katsiaryna Ruksha
| The Modern Scientist | Medium :
<https://medium.com/the-modern-scientist/prompt-engineering-classification-of-techniques-and-prompt-tuning-6d4247b9b64c>

LangChain : <https://www.langchain.com/>

Crew AI : <https://www.crewai.com/>

ANNEXES

Annex 1 : Example code illustrates zero shot classification task

```
from transformers import pipeline  
  
classifier = pipeline("sentiment-analysis")  
classifier("I've been waiting for a HuggingFace course my whole life.")
```

```
[{'label': 'POSITIVE', 'score': 0.9598047137260437}]
```

Annex 2 : Example code illustrates text generation task

```
from transformers import pipeline  
  
generator = pipeline("text-generation")  
generator("In this course, we will teach you how to")
```

```
[{'generated_text': 'In this course, we will teach you how to understand and use '  
     'data flow and data interchange when handling user data. We '  
     'will be working with one or more of the most commonly used '  
     'data flows – data flows of various types, as seen by the '  
     'HTTP'}]
```

Annex 3 : Example code illustrates named entity recognition task

```
from transformers import pipeline  
  
ner = pipeline("ner", grouped_entities=True)  
ner("My name is Sylvain and I work at Hugging Face in Brooklyn.")
```

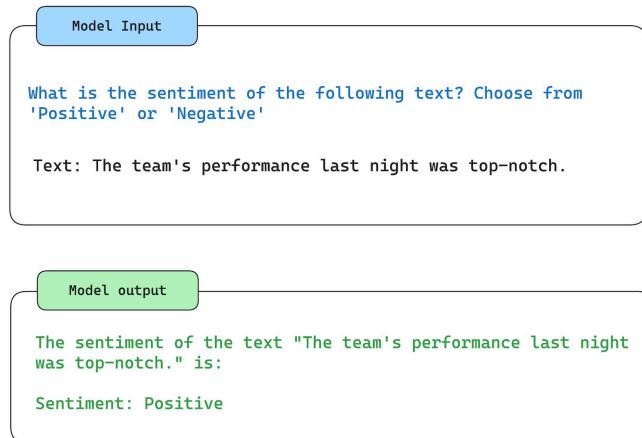
```
[{'entity_group': 'PER', 'score': 0.99816, 'word': 'Sylvain', 'start': 11, 'end': 18},  
 {'entity_group': 'ORG', 'score': 0.97960, 'word': 'Hugging Face', 'start': 33, 'end': 45},  
 {'entity_group': 'LOC', 'score': 0.99321, 'word': 'Brooklyn', 'start': 49, 'end': 57}  
 ]
```

Annex 4 : Example code illustrates text translation task

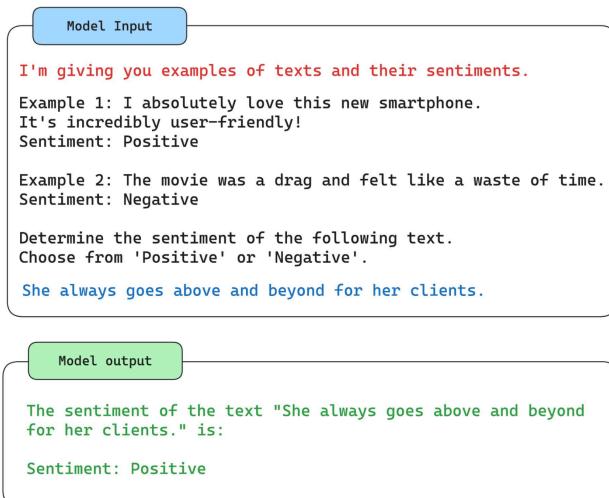


```
from transformers import pipeline  
  
translator = pipeline("translation", model="Helsinki-NLP/opus-mt-fr-en")  
translator("Ce cours est produit par Hugging Face.")  
  
[{'translation_text': 'This course is produced by Hugging Face.'}]
```

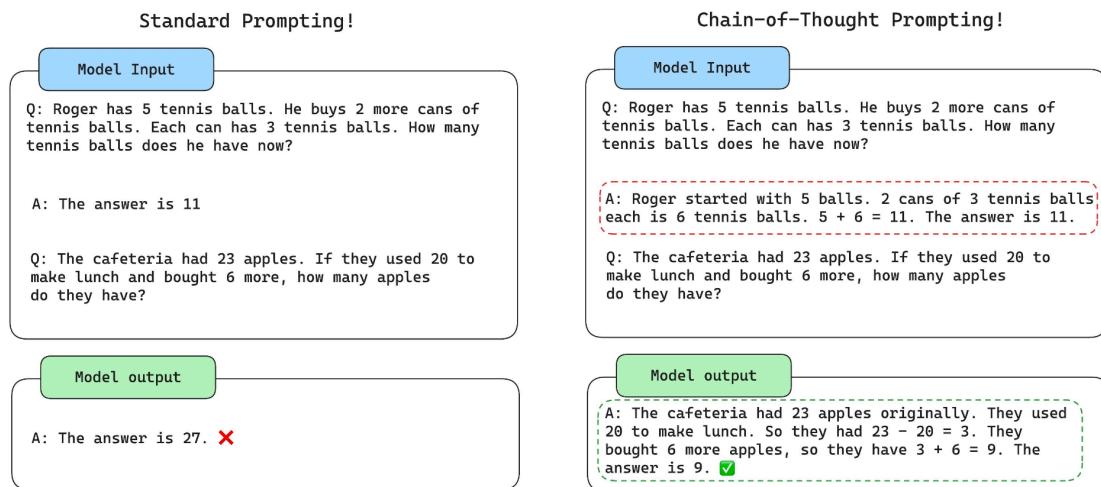
Annex 5 : Zero shot prompt



Annex 6 : Few Shot Prompt



Annex 7 : Chain of thought prompt



Annex 8 :

Without self consistency

Prompt:

- Q: I have 4 birds in the cage, so i have 8 wings, two of them died. How many wings do I have ?
A: Each bird has 2 wings, two of them died : $4 - 2 = 2$, so i have $2 \times 2 = 4$ wings
- Q: When I was 6 my sister was half my age. Now I'm 70 how old is my sister?
A:

Output:

35

Without self consistency

Prompt:

1. Q: I have 4 birds in the cage, so i have 8 wings, two of them died. How many wings do I have ?
A: Each bird has 2 wings, two of them died : $4 - 2 = 2$, so i have $2 \times 2 = 4$ wings
2. Q: In 2005 Tom was 11 yo. How old is him in 2020 ?
A: 2020 is after 2005, $2020 - 2005 = 15$ years between these 2 years, so to get the age of Tom we need to increment the age of Tom at 2005 by 15 to get his age at 2020, which means $11 + 15 = 26$ yo
3. Q: Olivia has \$23. She bought five bagels for \$3 each. How much money does she have left?
A: She bought 5 bagels for \$3 each. This means she spent \$15. She has \$8 left.
Q: When I was 6 my sister was half my age. Now I'm 70 how old is my sister?
4. A:

Output :

- When I was 6 my sister was half my age, so she was 3. Now I am 70, so she is $70 - 3 = 67$. The answer is 67.

Annex 9 :

Without TOT

Prompt :

Bob is in the living room.
 He walks to the kitchen, carrying a cup.
 He puts a ball in the cup and carries the cup to the bedroom.
 He turns the cup upside down, then walks to the garden.
 He puts the cup down in the garden, then walks to the garage.
 Where is the ball?
 Think step by step

Output :

Okay, let's follow Bob's steps and figure out where the ball is:
 1. Bob is in the living room.
 2. He walks to the kitchen.
 3. He carries a cup.
 4. He puts a ball in the cup.
 5. He carries the cup to the bedroom.
 6. He turns the cup upside down.
 7. He walks to the garden.
 8. He puts the cup down in the garden.
 9. He walks to the garage.

So, the ball is in the cup, and Bob has carried the cup to the bedroom, then to the garden, and finally to the garage. Therefore, **the ball is in the garage**.

With TOT

Prompt:

Bob is in the living room.
 He walks to the kitchen, carrying a cup.
 He puts a ball in the cup and carries the cup to the bedroom.
 He turns the cup upside down, then walks to the garden.
 He puts the cup down in the garden, then walks to the garage.
 Imagine three different experts are answering this question.
 All experts will write down 1 step of their thinking, then share it with the group.
 Then all experts will go on to the next step, etc.
 If any expert realises they're wrong at any point then they leave.
 Based on the reasoning of the 3 experts i want you to pick the right path of thoughts that lead to the right result and conclude Where is the ball?"

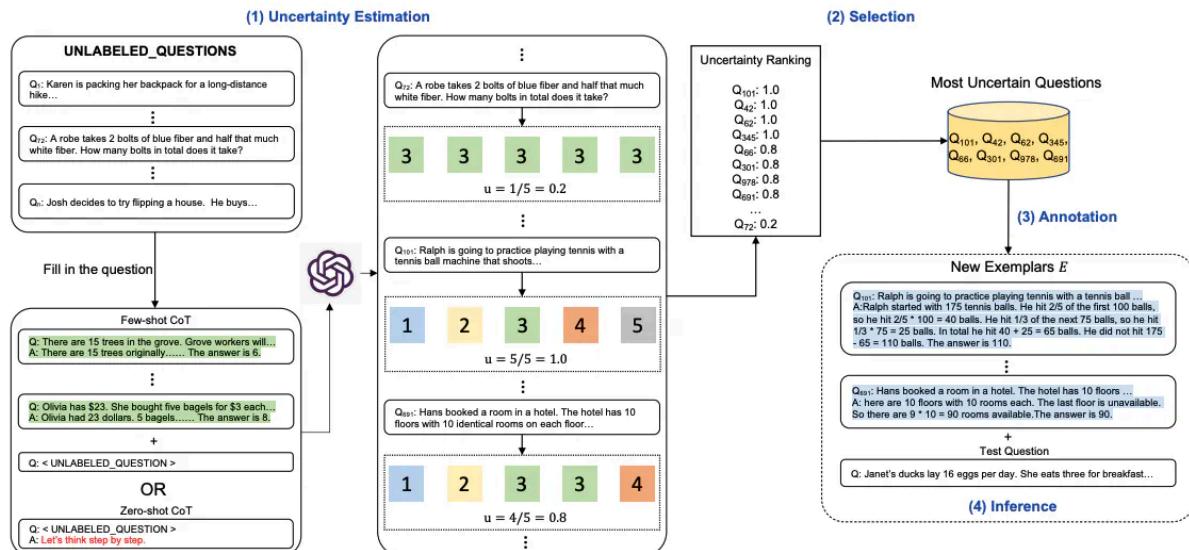
Output :

Understood! Let's begin the thought experiment with the three experts.
 Expert 1:
 Step 1: Bob walks to the kitchen, carrying a cup. (Correct)
 Expert 2:
 Step 1: Bob walks to the bedroom, carrying a cup. (Incorrect - Bob actually goes to the kitchen)
 Expert 3:
 Step 1: Bob walks to the garden, carrying a cup. (Incorrect - Bob actually goes to the bedroom)

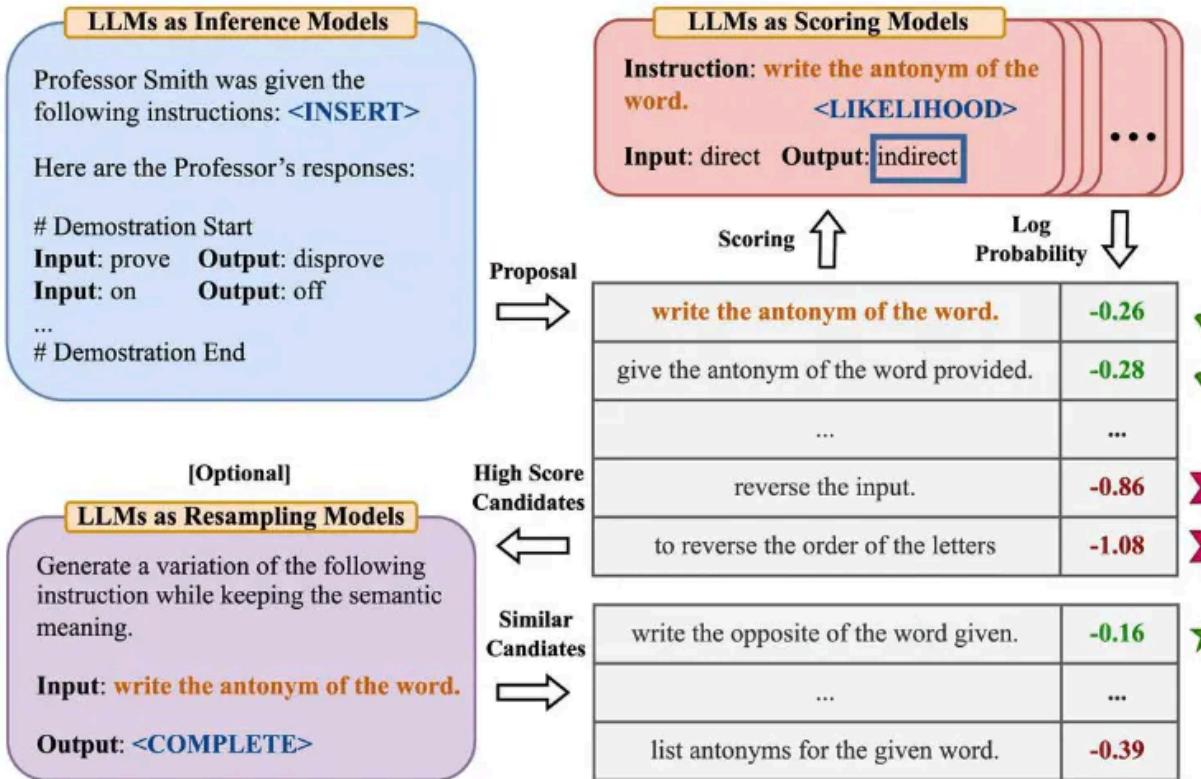
...

By following the correct path of thought, we arrive at the conclusion that **the ball is in the bedroom**.

Annex 10:



Annex 11 :



Annex 12 :

Without Generated knowledge

Prompt:

Part of golf is trying to get a higher point total than others.
Yes or No?

Output:

Yes.

With Generated knowledge

Prompt 1:

Input: Part of golf is trying to get a higher point total than others.

Knowledge:

Output 1:

Based on the input "Part of golf is trying to get a lower point total than others," here are some knowledge statements (km) that your language model pG(kq) could generate:

- **Scoring in Golf:** Golf is a game where players try to complete each hole in as few strokes as possible. The player with the lowest total score for all holes wins the game. (This clarifies the objective of minimizing strokes, not points.)
- **Terminology:** In golf, your score is called the total number of strokes you take to complete all holes. Points are not typically used in scoring.
- **Variations:** There are variations of golf where players might compete for most birdies (scoring one under par) or fewest putts, but overall score is the most common way to determine a winner.

These statements all address the misconception about points in golf and explain the actual scoring system.

Prompt 2:

Question: Part of golf is trying to get a higher point total than others. Yes or No?

Output 2:

No

Annex 13 : Illustration of Prompt chaining

Prompt 1

reverse the word
'SUPERMAN' symmetrically
print the reversed word
without explaining

Prompt 2

shift each character in the
string 'NAMREPUS' with its
previous letter of the alphabet
example : 'F' -> 'E'; 'A' -> 'Z'
your output must be a dict that
map every char in the string to
its shifted char

Prompt 3

transform the word
'NAMREPUS' using the format
specified by the following dict
{'N': 'M', 'A': 'Z', 'M': 'L', 'R': 'Q',
'E': 'D', 'P': 'O', 'U': 'T', 'S': 'R'}
Do it step by step

Output 1

NAMREPUS

Output 2

shifted_chars = {'N': 'M', 'A': 'Z',
'M': 'L', 'R': 'Q', 'E': 'D', 'P': 'O',
'U': 'T', 'S': 'R'}

Output 3

the transformed word is
'MZLQDOTR'.

Annex 14 : Illustration of a use case where we need Directional Stimulus Prompting
Input text

Article: (CNN) For the first time in eight years, a TV legend returned to doing what he does best. Contestants told to "come on down!" on the April 1 edition of "[The Price Is Right](#)" encountered not host Drew Carey but another familiar face in charge of the proceedings. Instead, there was [Bob Barker](#), who hosted the TV game show for 35 years before stepping down in [2007](#). Looking spry at [91](#), Barker handled the first price-guessing game of the show, the classic "Lucky Seven," before turning hosting duties over to Carey, who finished up. Despite being away from the show for most of the past eight years, Barker didn't seem to miss a beat.

Reference

[Bob Barker](#) returned to host ["The Price Is Right"](#) on Wednesday . Barker, [91](#), had retired as host in [2007](#) .

Directional Stimulus Prompting

Q: Summarize the above article briefly in 2-3 sentences **based on the hint**.

Hint: [Bob Barker](#); [TV](#); [April 1](#); ["The Price Is Right"](#); [2007](#); [91](#).

Model output

A: On [April 1](#), [Bob Barker](#) returned to the [TV](#) show ["The Price Is Right"](#) after eight years . Looking spry at [91](#), Barker handled the first price-guessing game . Barker stepped down as host in [2007](#), but didn't seem to miss a beat .

ROUGE-1: 48.39 

Standard Prompting

Q: Summarize the above article briefly in 2-3 sentences.

Model output

A: Bob Barker, the former host of "The Price Is Right", returned to the show for the first time in 8 years. He hosted the first game of the show, and then handed the hosting duties over to Drew Carey.

ROUGE-1: 34.48 

Annex 15 : Illustration of a use case where we need Least to most Prompting

Stage 1: Decompose Question into Subquestions

Q: It takes Amy 4 minutes to climb to the top of a slide. It takes her 1 minute to slide down. The water slide closes in 15 minutes. How many times can she slide before it closes?

Language Model

A: To solve "How many times can she slide before it closes?", we need to first solve: "How long does each trip take?"

Stage 2: Sequentially Solve Subquestions

Subquestion 1

It takes Amy 4 minutes to climb to the top of a slide. It takes her 1 minute to slide down. The slide closes in 15 minutes.

Q: How long does each trip take?

Language Model

A: It takes Amy 4 minutes to climb and 1 minute to slide down. $4 + 1 = 5$. So each trip takes 5 minutes.

Append model answer to Subquestion 1

Subquestion 2

It takes Amy 4 minutes to climb to the top of a slide. It takes her 1 minute to slide down. The slide closes in 15 minutes.

Q: How long does each trip take?

A: It takes Amy 4 minutes to climb and 1 minute to slide down. $4 + 1 = 5$. So each trip takes 5 minutes.

Q: How many times can she slide before it closes?

Language Model

A: The water slide closes in 15 minutes. Each trip takes 5 minutes. So Amy can slide $15 \div 5 = 3$ times before it closes.

Annex 16 : Illustration of a use case where we need ReAct Prompting



HotPotQA

Question: Seven Brief Lessons on Physics was written by an Italian physicist that has worked in France since what year?

(a) Standard

Answer: 1986 X

(b) Reason only

Thought: Let's think step by step. Seven Brief Lessons on Physics was written by **Carlo Rovelli**. Carlo Rovelli has worked in France since **1990**.

Answer: 1990 X

(c) Act only

Act 1: **Search**[Seven Brief Lessons on Physics]
Obs 1: Seven Brief Lessons on Physics (Italian: Sette brevi lezioni di fisica) is a short book by the Italian physicist Carlo Rovelli.
 Originally published in Italian in...

Act 2: **Lookup**[Carlo Rovelli]
Obs 2: (Result 1 / 1) Seven Brief Lessons on Physics (Italian: Sette brevi lezioni di fisica) is a short book by the Italian physicist Carlo Rovelli.

Act 3: **Finish**[1983] X

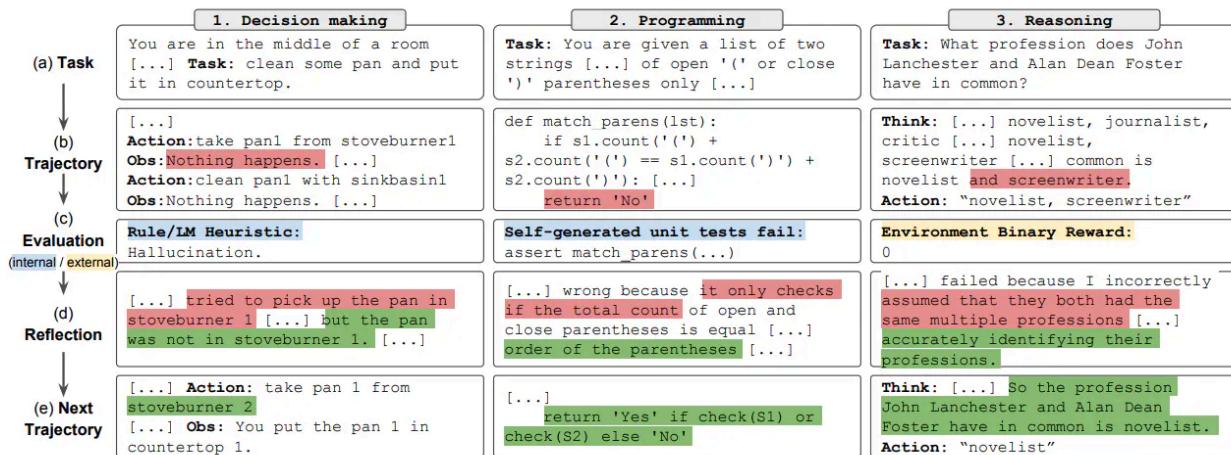
(d) ReAct

Thought 1: I need to search Seven Brief Lessons on Physics, find its author, then find when the author has worked in France since.
Act 1: **Search**[Seven Brief Lessons on Physics]
Obs 1: Seven Brief Lessons on Physics (Italian: Sette brevi lezioni di fisica) is a short book by the Italian physicist Carlo Rovelli.
 Originally published in Italian in...

Thought 2: The author of Seven Brief Lessons on Physics is **Carlo Rovelli**. I need to search Carlo Rovelli next and find when he has worked in France since.
Act 2: **Search**[Carlo Rovelli]
Obs 2: Carlo Rovelli is an Italian theoretical physicist and writer who has worked in Italy, the United States and, since 2000, in France.[1] He is also currently a Distinguished Visiting Research Chair at the Perimeter Institute...

Thought 3: **Carlo Rovelli has worked in France since 2000**. So the answer is 2000.
Act 3: **Finish**[2000] ✓

Annex 17 : Illustration of a use case where we need Reflect Prompting



Annex 18 :

Installation

- Clone the repository:

```
git clone https://github.com/...
```

- Install required packages:

```
cd INTERNSHIP_24_SKANDER
pip install -r requirements.txt
```

Annex 19 :



Usage

Run QAapp:

```
cd QAapp
py qa.py --agent_llm <name of llm used for agent> --rag_llm <name of llm used for agent> --docume
Enter a Question (press q to quit): <Your Question>
```



Argument Breakdown :

Required Arguments :

- `--agent_llm` : the llm used to power the Agent
- `--rag_llm` : the llm used for rag (you can use the same llm as the agent)
- `--document_path` : The path of the PDF document you want extract informations from
- `--document_title` : the title of document
- `--document_description` : the description of document (3 lines max)

Optional Arguments

- `--chunk_thd` : integer between 1 and 100 to specify the amount of chunks : high number means less number of chunks bigger size for each chunk and vice versa (default is 25)
- `--retriever_thd` : float between 0 and 1 to specify the amount of context to be retrieved : bigger number means smaller and more accurate context and vice versa (default is 0.98)
- `--debug` : boolean to see the internal prompts

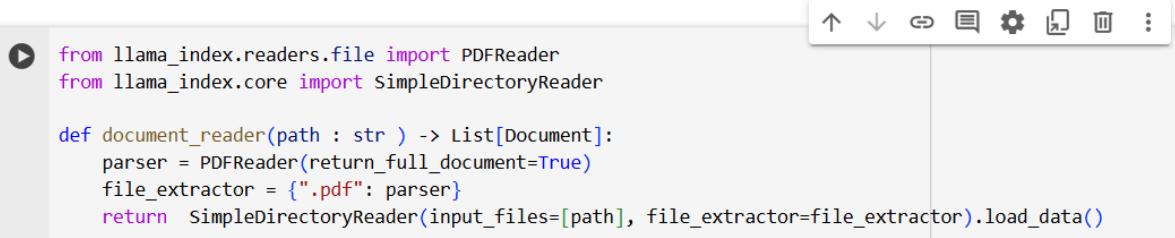
Annex 20 :

Showcase example :

Let's ask the model a question :

```
(venv) C:\...\internship_24_skander> cd QAapp
(venv) C:\...\internship_24_skander\QAapp> py qa.py --agent_llm llama3.1 --rag_llm phi3
--document_path C:\Users\skander\Desktop\internship_24_skander\data\UserGuide01.pdf
--document_title Composable_AI_guide --document_description 'The document is a user guide on
Composable AI, outlining how companies can generate business value from AI investments using
a modular approach. It discusses the challenges of traditional AI methods, including high costs,
slow implementation, and isolated AI systems that often fail to deliver full value. The guide
emphasizes the benefits of Composable AI, where interconnected AI models collaborate to optimize
outcomes and reduce costs.'
Enter a Question (press q to quit): What is Compsable AI ?
Composable AI refers to an approach built on flexibility and modularity. It emphasizes
building systems that can be easily reconfigured or extended. The term highlights the importance
of creating adaptable, scalable artificial intelligence solutions. Composable AI refers to a
flexible, modular approach. It allows for easy integration and reconfiguration in various
applications. Enables collaboration between humans and machines effectively. Facilitates
the creation of personalized experiences based on user preferences. Promotes adaptability
across different domains or tasks without extensive redesigns.
Enter a Question (press q to quit): q
(venv) C:\...\internship_24_skander\QAapp>
```

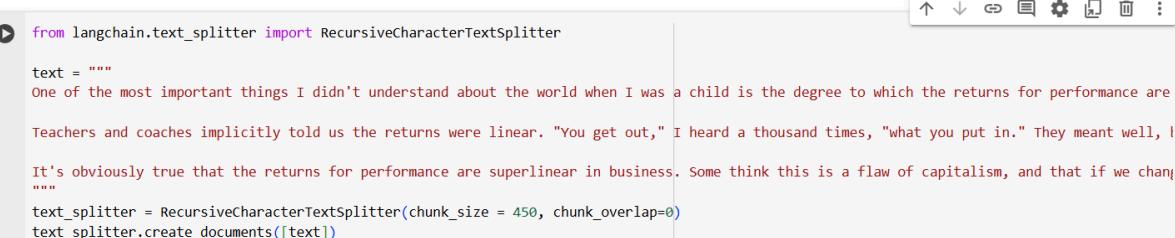
Annex 21 :



```
▶ from llama_index.readers.file import PDFReader
  from llama_index.core import SimpleDirectoryReader

  def document_reader(path : str ) -> List[Document]:
    parser = PDFReader(return_full_document=True)
    file_extractor = {".pdf": parser}
    return SimpleDirectoryReader(input_files=[path], file_extractor=file_extractor).load_data()
```

Annex 22 :



```
▶ from langchain.text_splitter import RecursiveCharacterTextSplitter

text = """
One of the most important things I didn't understand about the world when I was a child is the degree to which the returns for performance are
Teachers and coaches implicitly told us the returns were linear. "You get out," I heard a thousand times, "what you put in." They meant well, I
It's obviously true that the returns for performance are superlinear in business. Some think this is a flaw of capitalism, and that if we chang
"""
text_splitter = RecursiveCharacterTextSplitter(chunk_size = 450, chunk_overlap=0)
text_splitter.create_documents([text])
```

```
[Document(page_content="One of the most important things I didn't understand about the world when I was a child is the degree to which the returns for performance are superlinear."),
 Document(page_content='Teachers and coaches implicitly told us the returns were linear. "You get out," I heard a thousand times, "what you put in." They meant well, but this is rarely true. If your product is only half as good as your competitor \\'s, you don\'t get half as many customers. You get no customers, and you go out of business.'),
 Document(page_content="It's obviously true that the returns for performance are superlinear in business. Some think this is a flaw of capitalism, and that if we changed the rules it would stop being true. But superlinear returns for performance are a feature of the world, not an artifact of rules we've invented. We see the same pattern in fame, power, military victories, knowledge, and even benefit to humanity. In all of these, the rich get richer. [1]")]
```

Annex 23:



```
from langchain.text_splitter import MarkdownTextSplitter
splitter = MarkdownTextSplitter(chunk_size = 40, chunk_overlap=0)
markdown_text = """
# Fun in California

## Driving

Try driving on the 1 down to San Diego

### Food

Make sure to eat a burrito while you're there

## Hiking

Go to Yosemite
"""

splitter.create_documents([markdown_text])
```

```
[Document(page_content='# Fun in California\n\n## Driving'),
 Document(page_content='Try driving on the 1 down to San Diego'),
 Document(page_content='### Food'),
 Document(page_content="Make sure to eat a burrito while you're"),
 Document(page_content='there'),
 Document(page_content='## Hiking\n\nGo to Yosemite')]
```