Study of usage of Large Language Model in industrial context

Auteur : Mokeddem, Samy
Promoteur(s) : Ernst, Damien
Faculté : Faculté des Sciences appliquées
Diplôme : Master : ingénieur civil électricien, à finalité spécialisée en Neuromorphic Engineering
Année académique : 2023-2024
URI/URL : http://hdl.handle.net/2268.2/20253

Université de Liège
Faculty of Applied science

Study of usage of Large Language Model in industrial context

Mokeddem Samy - S184315
samy.mokeddem@student.uliege.be

Academic year 2023-2024

June 9, 2024

Acknowledgments

First of all, I would like to thank Lize Pirenne, with whom I have worked in close collaboration over the past year. I wish him good luck for the for the rest of her PhD. thesis. I would also like to thank my promoter, Professor Damien Ernst, for his help and for introducing me to the fascinating world of research. I should also thank Sebastien Renaud, Philipe Drugmand, Eric Tang and Didier Fuss for their feedback, their help and and for carefully reviewing this work. Many thanks to teams of Haulogy that provide me a serene and stimulating work environment. Finally, I would like to thank those persons the closest to me: my friends and, of course, my parents for provide me support and encourage me during this work and more general my all studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The rapid advancements in artificial intelligence (AI) and machine learning (ML) have ushered in an era where large language models (LLMs) like GPT-4[1], Gemini[2], Llama2[3], and their successors are at the forefront of technological innovation. These models, designed to understand and generate human language, have demonstrated unprecedented capabilities across a wide array of applications, from natural language processing (NLP) tasks to complex decision-making processes. The LLMs have a huge transformative potential in industrial contexts, where they promise to revolutionize the operations, enhance productivity, and drive significant cost savings [4].

This work explores the integration and utilization of LLMs in industrial settings, with a specific focus on the case of Haulogy[1], a growing Belgium IT enterprise for the energy sector. By examining this practical example, we aim to investigate the practical applications of LLMs in industrial contexts, identify the challenges associated with their deployment, and propose solutions to overcome these challenges.

In the first part of this work, we will formalize EnergyComm, a software solution developed by Haulogy, to better understand its functionality. Based on this formalization, we will discuss potential improvements that LLMs can bring. One promising idea is the retrieval augmented generation (RAG) framework [5], which enables interaction with LLM chat models using a personalized knowledge dataset.

The second part of this work will define and test this RAG framework. We will discuss the practical applications of this framework in an industrial context, evaluate its effectiveness and potential benefits but also the weaknesses that can stop its deployment.

In the third and final part, this work proposes an enhancement to closed-domain question answering (CQA) with large language models (LLMs) by introducing several methods to tackle the task of sentence-level rationale extraction from generated answers. The goal of this task is to identify the smallest set of sentences within the context necessary to correctly answer a given question, these sentences are referred to as rationales. Accurate prediction of these rationales enhances the explainability of the LLM-generated answers. To evaluate the proposed methods, two new elements are introduced: an enhanced CQA dataset with custom annotations and a new metric for assessing the relevance of the predicted rationales. The dataset includes highlighted sentences within the context that constitute the rationales of answer to the question, while the new metric, termed the IoU score, measures the overlap (Intersection over Union) between the predicted and expected rationales.

Through this structured approach, this work aims to bridge the gap between the theoretical capabilities of LLMs and their practical implementation in industries, providing insights that can guide future research and industrial strategies.

1 `https://haulogy.net/fr/accueil/`

Chapter 2

EnergyComm: Formalization and potential integrations of LLMs

2.1 Introduction

EnergyComm is an IT product that enables energy suppliers and BRP to communicate with the energy markets of different countries through a single unified platform. In fact, the distribution network operators (DNOs), the energy suppliers, the balancing responsible parties (BRPs) and sometimes the transport system operators (TSOs) of each country communicate through numerical messages. The exchange format, content, and sending modality of these numerical messages are different for each nation. By simplicity, in the rest of this report this exchange platform will be refereed as the **Clearing House** (CH). Since each country has its own rules, each market differs and supporting them is a complex and time-consuming task. In this context, EnergyComm offers a communication interface that accommodates the specificities of various supported national markets (currently Belgium, France, Netherlands). This adaptability is possible trough adaptive module known as the market adapter by country. While the core functionality of EnergyComm remains consistent across all countries, the market adapter is tailored specifically for each individual country.

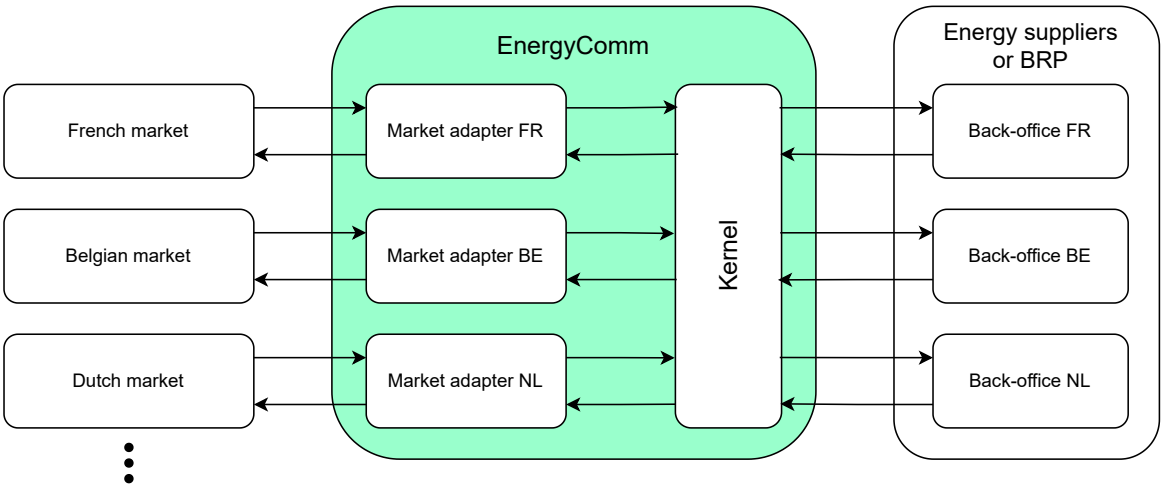


Figure 2.1: EnergyComm concept

On the other hand, while EnergyComm manages the majority of messages received by the Clearing House correctly, a small portion of messages is rejected by EnergyComm. These rejected messages are addressed by the operating team, which carries out numerous manual corrections in order to reduce the number of rejected

messages to treat. These tasks involve tasks such as reformatting poorly formatted messages, analyse the rejected messages, among others.

2.2 Markets and Clearing House

As mentioned earlier, each country has its own communication rules governing the interactions among various actors in the energy market. Despite this, there is a common structure that will be explained in this section. The energy market comprises the following key actors:

- **TSO (Transmission System Operator)**: Responsible for transporting energy to various access points of the DNOs.
- **DNO (Distribution Network Operator)**: In charge of physically transporting energy from the transmission grid to the end consumers. Each DNO is responsible for a specific geographical region.
- **Energy Suppliers**: Responsible for negotiating contracts with end consumers and providing energy to them. To supply energy, the energy suppliers must communicate with the DNOs and TSOs to obtain the energy consumption data of the access points they are responsible for. They then communicate this data to its BRP.
- **BRP (Balancing Responsible Party)**: Responsible for maintaining the appropriate balance between energy consumption and production for the access points they oversee by purchasing or selling energy on the market from various producers.
- **ISO (Independent System Operator)**: Ensures the security of the entire electricity power system. It manages the global balance over the network by overseeing the production and consumption of electricity. For example, ISOs can impose penalties on BRPs that create imbalances and, conversely, offer incentives to BRPs that help mitigate imbalances.
- **CHP (Clearing House Platform)**: Defines communication protocols and standards. CHPs are not present in every country; for example, they are not present in France.
- **Regulator**: Its primary role is to establish and approve electricity market rules. Additionally, it investigates suspected cases of abuse and exercises control over the prices of products and services in monopolistic situations, such as distribution network fees.

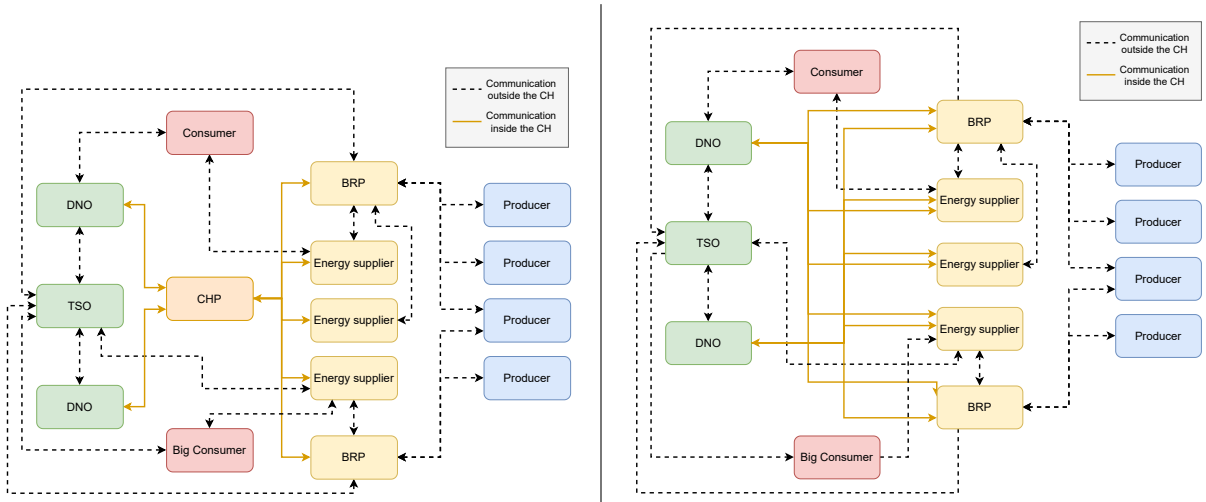
Each of these actors communicate with each other through a series of numerical messages. These message exchanges, involving the entities within a given country, occur within a virtual space known as the Clearing House (CH). In the remainder of the report, any reference to a *Clearing House message* or *CH message* refers to messages occurring within this virtual space.

Within the structure of this energy market, the DNO assumes a central role in the communication process. This central role is attributed to the DNOs' neutrality, as they maintain a stable number of clients due to regional allocations. DNOs are responsible of all access points within their respective regions, effectively holding a natural monopoly in this regard. Conversely, energy suppliers compete with one another, vying to secure as many customers as possible. As a result, regulatory decisions have mandated that communication between energy suppliers is channeled through the DNO.

Both energy suppliers and BRPs communicate directly with DNOs, and occasionally with TSOs. In this configuration, every energy supplier is required to engage in communication with multiple DNOs across the market. The primary objective of EnergyComm is to facilitate communication between energy suppliers, BRPs and all these DNOs through a unified interface.

However, in certain countries, a CHP (e.g., Atrias in Belgium) aims to standardize these communication transactions. This entity defines a series of protocols that each DNO must adhere to in order to communicate through it. Additionally, this CHP serves as a liaison between all the DNOs and all the energy suppliers, ensuring smoother interactions between the two parties.

The physical structure of an energy market is presented in the fig. 2.3. The communication structure of an energy market is shown in the Fig. 2.2.



(a) With CHP (b) Without CHP

Figure 2.2: Communication structure of a energy market

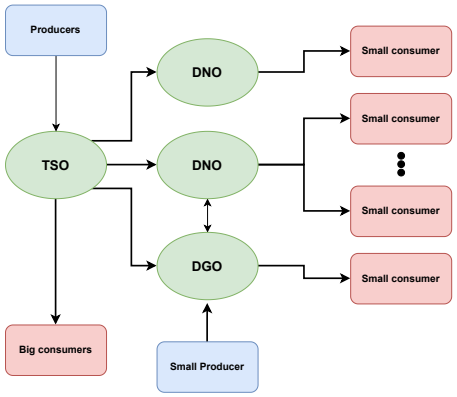


Figure 2.3: Physical structure of a energy market

2.3 Presentation of EnergyComm

EnergyComm is an independent module of the suite of softwares commercialized by Haulogy. EnergyComm is one of the two main components of the software Haugazel. Haugazel provides a solution to manage the billing of customers for energy suppliers. EnergyComm serves as the Clearing House gateway of Haugazel and can also be used independently. In the context of technological interfaces or systems that facilitate the exchange of information, data, and transactions between the various participants, EnergyComm is the Clearing House gateway. Specifically, within the Clearing House, EnergyComm facilitates the exchange of Clearing House messages (CH messages) between energy suppliers or BRPs and the DNOs, TSOs or CHPs (such as Atrias in Belgium). This gateway acts as a bridge, enabling seamless communication, data transmission, and interaction among these diverse actors and the Clearing House infrastructure.

Before delving into the explanation of the different modules of EnergyComm, it is essential to provide a definition for a CH message and a Kernel message.

A CH message is a collection of data fields organized in a specific format. This format is established by the CHP, if it exists, or by the GRDs if the platform is absent. Furthermore, there are various types of CH messages, each serving a specific purpose. While each CH message within a particular CH adheres to a common structure, each message type possesses its own set of data fields present in the message.

On the other hand, to accommodate the specific requirements of various energy markets, EnergyComm incorporates a collection of generic energy messages, commonly referred to as Kernel messages. These generic messages comprise multiple fields designed to encompass all the necessary data fields for the CH messages of the supported CH. Moreover, a CH message is converted into a set of Kernel messages (Eq. 2.4). So there is no bijection between the CH message space (M_{CH}) and the Kernel message space (M_K). These Kernel messages are formatted in XML[1]. The use of XML facilitates structured and flexible data organization. Another advantage of the XML format is that the majority of CH message exchanged on different CH is also in XML format, simplifying the conversion process. Therefore, a Kernel message can be described as a set of data fields presented in XML format.

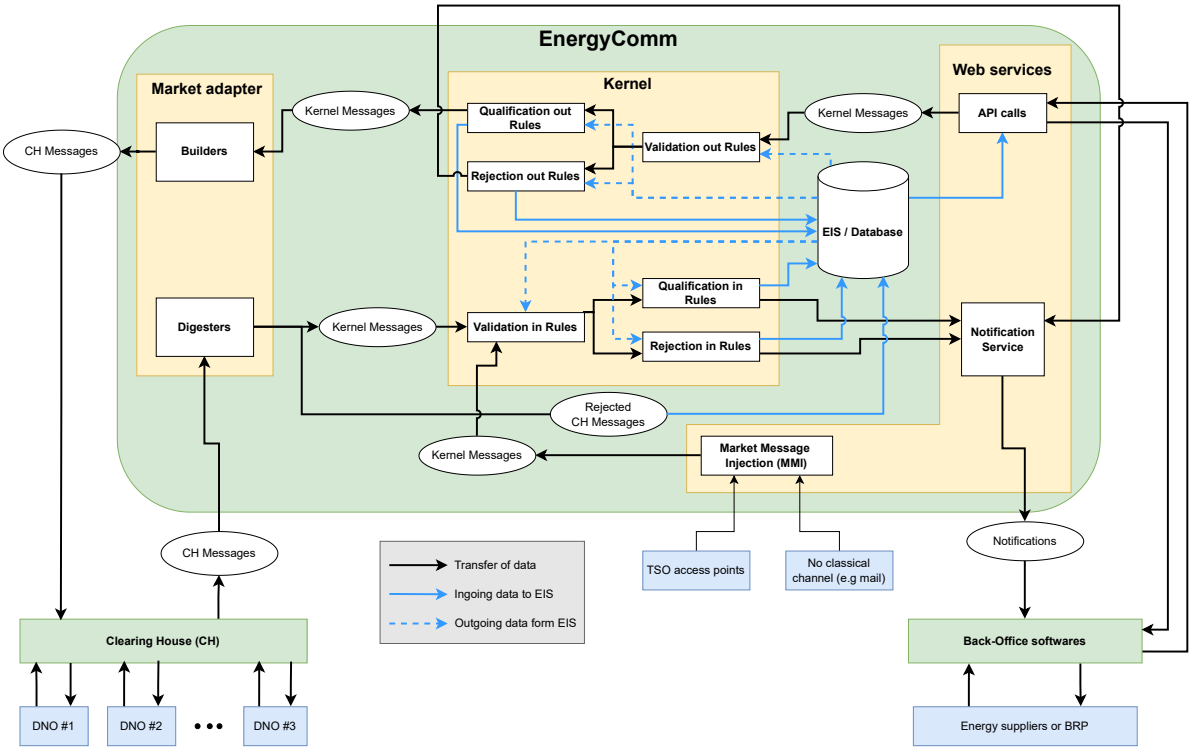


Figure 2.4: EnergyComm Diagram

As shown in Figure 2.4, the EnergyComm product consists of various modules. It is important to note that the figure in Figure 2.4 represents a simplified version of EnergyComm, showing only the principal modules for the sake of clarity. The sub-modules will be presented and described in more specific sections. Additionally, there are numerous data backup operations, which occur almost before each data operation. However, these backup operations are not indicated in the figure also for the sake of clarity also.

1 XML (Extensible Markup Language) is a versatile and widely used format for structuring and storing data in a hierarchical and human-readable way. It consists of tagged elements that provide a standardized approach to represent and exchange information, making it a fundamental technology for data interchange and storage across various applications and platforms.

Now, let's proceed to describe the principal modules of EnergyComm:

- **The Market adapter** is the module responsible for facilitating communication with the market. This module operates in two primary directions: the incoming direction and the outgoing direction.

  In the incoming direction, the Market adapter is tasked with converting the CH messages received from the CH into one or more Kernel messages. These Kernel messages are then processed within the system.

  Conversely, in the outgoing direction, the Market adapter is responsible for transforming Kernel messages generated by the web services into CH-specific message formats, which could include XML, CSV, EDI, and others.

  Additionally, the Market adapter manages the transport layer, which, in the context of the energy market, refers to the infrastructure and protocols that ensure secure and efficient data exchange between various market participants and systems. This involves establishing communication with the relevant actors using different protocols in place, such as Web Services (WS), secure file transfer protocols (SFTP), and more. It is worth noting that the quantity of messages received by the Market adapter significantly exceeds the number of messages it sends. This is a result of the CH configuration, which places the energy supplier in a relatively passive role, primarily receiving data and information from the CH.

  The Market adapter comprises several sub-modules, but this section will focus exclusively on the two primary sub-modules: the Digesters and the Builders. Detailed descriptions of the other sub-modules can be found in Section 2.4.

  - **The Digesters** transform a CH message into a set of Kernel messages (EnergyComm breaks down the message into the most unitary concepts possible). The size of this set is variable and can contain only one message or several messages depending of the type of the CH message processed. The Digesters are used for any messages pushed by the CH to EnergyComm. Additionally, the Digesters has also a role of syntax verification. During the transformation of the message, the format and content of each field in the message pass some checks. If the content is not in the correct format (e.g., the Digester expects an integer format for some fields), the message is promptly rejected, and its content is stored in the database.
  - **The Builders** transform the standard Kernel message into a CH message. Conversely to the Digester the Builder transform a kernel into a single CH so there is a bijection between the CH and the kernel message transformed by the Builders.

- **The Kernel** is an ensemble of *rules engine* (referenced in [6] and [7]), each having its own set of rules. To begin with, a rule is a declarative statement or condition that outlines how the system should behave based on certain input. These rules are used with a kernel of rules algorithm to automate the decision process. Indeed, each rule establishes a set of criteria or conditions, and when these conditions are met, they trigger specific actions or a series of actions. Moreover, the data on which these rules operate is referred to as "facts".

  Before delving into the specifics of the Kernel's rules engines, let us first define what a rules engine is. A rule engine can be likened to an algorithm that executes the rules in a deterministic order. It combines a set of facts, which are inserted into the system, with its own rule set to arrive at conclusions that lead to the triggering of one or more actions.

  A set of rules can be processed either in a sequential mode or in parallel mode. When processed in sequential mode, each rule within the set follows a specific order. For instance, Rule n in a set of rules R depends on the outcomes of the $n-1$ preceding rules. Certain rules may be skipped based on the results of the previous rules.

  Conversely, rules can be processed in parallel mode, where the order of execution is not significant. In this scenario, all rules are independent, and even though they may be executed sequentially by the kernel, no rules can be skipped regardless of the results of other rules in the set. In the case of EnergyComm, rules are executed in parallel mode.

Now the different set of rules will be described but not formalized. This formalization will be done in the section 2.4:

- Validation in Rules: The purpose of validation within the rules framework is to ensure the coherence of messages originating from the CH before they are transmitted to the Back-Office software [2]. If any of the rules in this set are violated, the message is rejected; otherwise, it is accepted. The accepted messages subsequently undergo processing through the Qualification in Rules, while the rejected messages are directed to the Rejection in Rules for processing.
- Qualification in Rules: process the messages accepted by the Validation in Rules. The messages are stored in a table of the database and the other tables of the database are updated with the relevant information in the message. Then a qualification notification is transmitted to the Back-Office. Additionally, some message cannot be processed immediately because some other messages are expected. Indeed, some messages need other messages to be correctly processed, and it may happen that these messages come only after a while. So to mitigate this issue the messages expecting other messages are quarantined in the database until the incoming of the expected message.
- Rejection in Rules: process the messages rejected by the Validation in Rules. The system stores these rejected messages in the database along with their corresponding error codes. Additionally, a rejection notification is transmitted to the Back-Office. Within the Rejection in Rules set, specific messages may be quarantined for reasons similar to those explained earlier.

  It is crucial to emphasize that Validations in Rules are executed in parallel mode, as mentioned previously. The parallel mode implies that all validation rules are executed irrespective of the outcomes of other Validations in Rules. Consequently, it is possible for a rejected message to be associated with multiple error codes.
- Validation Out Rules: Ensure that messages and requests from suppliers adhere to the specifications and regulations of the target market. If any rule from this set is violated, the message is rejected. Accepted messages are then processed by the Qualification Out Rules, while rejected messages are handled by the Rejection Out Rules.
- Qualification out Rules: process the messages accepted by the Validation in Rules. The messages are stored in a table of the database and the other tables of the database are updated with the relevant information in the message. For this set of rules, also some messages are placed in quarantine for the same reason that explained before.
- Rejection out Rules: process the messages rejected by the Validation out Rules. The rejected messages are stored into the database with the corresponding error codes. Also, sometimes a notification of rejection is sent to the Back-Office. It is important to notice that the Validations out Rules are all evaluated, regardless of the results of the other Validations out Rules. Thus, it is possible that many error codes are linked to the rejected message. For this set of Rules, some messages are also placed in quarantine for the same reasons as explained before.

2 The term "Back-Office software" refers to the software utilized by Energy suppliers or BRPs for conducting Back-Office tasks. For instance, Haugazel is an example of Back-Office software developed and marketed by Haulogy. Additionally, a Back-Office consists of administrative operators responsible for overseeing various financial operations, including payment verification, invoicing, and payment reminders, among others.

All the rules applied by the Kernel have been documented in a file called *KernelRules* [8]. The Kernel and its rules are one of EnergyComm's significant assets and are formulated by EnergyComm's analysts. They result from client business requirements and CH specifications.

- The **Energy Information Store** or **EIS** is a database whose purpose is to provide insights into what information has been transmitted by the CH and the BackOffice for the different access points. To do that EIS stores the Kernel message and all the information they contain. The EIS store also the history of CH messages receives by the CH and also sent to it. Moreover, this database store the links between the CH-message and the kernel messages generated. So, by analyzing the database tables, we can observe the various actions taken by EnergyComm.
- Finally, the last module is the web services interface that enables the Back-Office softwares of EnergyComm users to communicate with EnergyComm through web services like API calls[3]. These web services allow users to perform various actions on the database and/or interact with the market. Moreover, the web service contains a special service called **Market Messages Injection** or **MMI**. The MMI enables bypassing the Market Adapter and sending messages directly to the Kernel. The purpose of this web service is to integrate access points from which information doesn't come through CH, such as TSO access points or when DNOs send information about access points via email.

2.4 CH message from CH to Back-Office

We formalize in this section the path of a message coming from the CH and going through EnergyComm to the EIS. This message path is defined by the main four blocks described below. Moreover, a diagram of this process is shown of the Fig. 2.5.

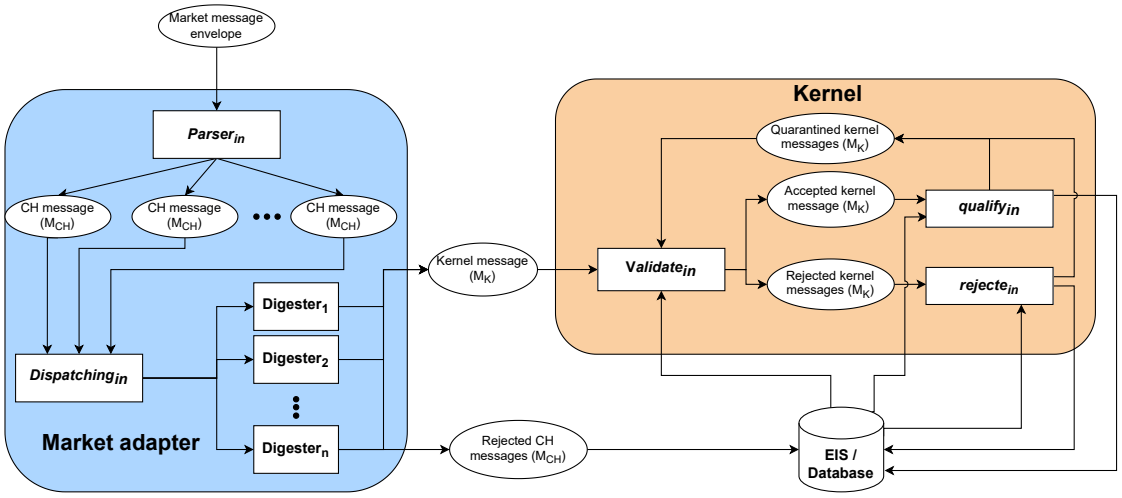


Figure 2.5: CH message process trough EnergyComm modules

1. Firstly, an envelope of messages arrives at EnergyComm. Indeed, the CH messages arrive in packets and are encapsulated within an envelope. This envelope is then split into the different messages it contains. This splitting is done by the parser module that can be modeled as follows:

$$\{m_{CH}^1, m_{CH}^2, \ldots, m_{CH}^n\} = Parser_{in}(e) \tag{2.1}$$

where $e \in E$ is a envelope of message, $m_{CH}^j \in M_{CH}$ is the j-th CH message is contained in the envelope e.

3 An API call, or Application Programming Interface call, is a request made by one software application to another in order to retrieve or exchange data and functionality. These calls serve as a means for different software systems to communicate and interact, allowing them to share information and perform tasks seamlessly. API calls are essential in modern software development, enabling the integration of various services and applications, and facilitating the exchange of data that enhances the functionality and capabilities of software systems.

2. The CH message arrives and is transformed into one or more Kernel messages. This transformation is carried out by the Market adapter module, specifically by the various Digesters within the module. Depending on the type of CH message received, the CH message is pushed to the appropriate Digester. This dispatching of the message is done by the Dispatcher. Additionally, the syntax of the CH message is verified by the builder and the CH message can be rejected if some fields are in the wrong format for example. So the Market adapter module can be conceptualized as the following functions:

$$\{l, s\} = Adapter_{in}(m_{CH}) \equiv Digester_i(m_{CH,i}) \tag{2.2}$$

$$\{l, s\} = Digester_i(m_{CH,i}) \tag{2.3}$$

$$i = Dispatch_{\text{in}}(m_{CH}) \tag{2.4}$$

With $m_{CH} \in M_{CH}$ represents a CH message, $l \in \{valid, rejected\}$ is the label of the CH-message and $s \in S$ is a set of Kernel messages. Indeed $s = \{m_K^1, m_K^2, \ldots, m_K^n\}$ and $s \subset M_K^n$. On the other hand, i is the index of the Digester for the CH messages of type i. $m_{CH}^i \in M_{CH}^i$ is a CH message of type i. Additionally, $M_{CH}^i \subset M_{CH}$.

3. Next, the transformed CH message, now an Kernel message in XML format, is processed by the Kernel. As explained previously (cfr. section 2.3), The Kernel applied a collection of rules to it.

   First, the Validation in Rules are applied to the incoming message. If any of the Validation in Rules are not met, the message is labeled as rejected along with the associated errors code and error comment. Conversely, if all the Validation in Rules are satisfied, the message is accepted.

   Following the Validation in Rules, the Qualification in Rules are applied to the message if it is accepted. Conversely, if the message is rejected, the Rejection in Rules are evaluated to ensure its proper handling. Additionally, certain Qualification in Rules and Rejection in Rules check whether certain expected messages have arrived before the final processing of the current message. If these expected messages have not yet arrived, the message is quarantined in the database. On the other hand, certain Qualification in Rules and Rejection in Rules have actions that involve replaying quarantined messages.

   The incoming messages are processed by the $Kernel_{in}$ function as follows:

$$\{l,\ s_a,\ s_{err},\ c\} = Kernel_{in}(m_k,\ h_m) \tag{2.5}$$

   where the input $m_k \in M_k$ a kernel message and $h_m \in H_m$ the history of messages linked to m_k. So $h_m \subset M_k^n$ is a set of n kernel message. $y = \{l,\ s_a,\ s_{err},\ c\}$ is the output of the function with l the label of the message, s_a the set of actions to process it, s_{err} the set of error codes and $c \in C$ as comments. It is noteworthy that if $l =$ accepted, then $s_{err} = c = \emptyset$.

   This function can be further subdivided into more specific functions that will be described below:

$$\{l,\ s_{err},\ c\} = Validate_{\text{in}}(m_k,\ h_m) \tag{2.6}$$

   where $m_k \in M_k$ represents a kernel message, and $h_m \in H_m$ denotes the history of messages associated with m_k. Thus, $h_m \in M_k^n$ is a set of n kernel messages. The output space, $y = \{l,\ s_{err},\ c\}$, comprises $l \in \{\text{rejected, accepted}\}$ as the label of the message, s_{err} as the set of error codes, and c as comments. It is noteworthy that if $l =$ accepted, then $s_{err} = c = \emptyset$. .

$$\{l,\ s_a\} = Qualify_{\text{in}}(m_k,\ h_m) \tag{2.7}$$

   where m_k, h_m have the same significance as in the previous equation. The output is $y = \{l, s_a\}$ with l representing the label of the message, s_a indicating the set of actions to process it.

$$\{l,\ s_a,\ s_{err},\ c\} = Reject_{\text{in}}(m_k,\ s_{err},\ h_m) \tag{2.8}$$

where m_k, h_m have the same significance as in Eq. 2.6 and s_{err} is the set of error codes linked to the kernel message (m_K). The output is $y = \{l,\ s_a,\ s_{err},\ c\}$ with l representing the label of the message, s_a the set of actions to process it, s_{err} denoting the set of error codes, and c specifying the comments.

With the sub-functions defined, the $Kernel$ function can be expressed as :

$$Kernel(m_k,\ h_m) \equiv \begin{cases} Qualify_{in}(m_k,\ h_m)\,, & \text{if } l = accepeted \\ Reject_{in}(m_k,\ s_{err},\ h_m)\,, & \text{if } l = rejected \end{cases}$$

with $\{l,\ s_a,\ s_{err},\ c\} = y = Validate_{in}(m_K,\ s_{err},\ h_m)$.

4. The rejected messages blocked by the EnergyComm rules engine are saved in the database, awaiting processing by the EnergyComm operations team. The correction method of the operations team is described in detail further, and it can be modeled by the function of the Eq. 2.16. This general function will be detailed and divided into sub-functions in Section 2.6. Moreover, this function and sub-function is shown on the Fig. 2.7.

2.5 Kernel message from Back-Office to CH

We formalize in this section the path of a message coming from the Back-Office and going through EnergyComm to the CH. This message path is defined by the main four blocks described below. Moreover, a diagram of this process is shown of the Fig. 2.6.

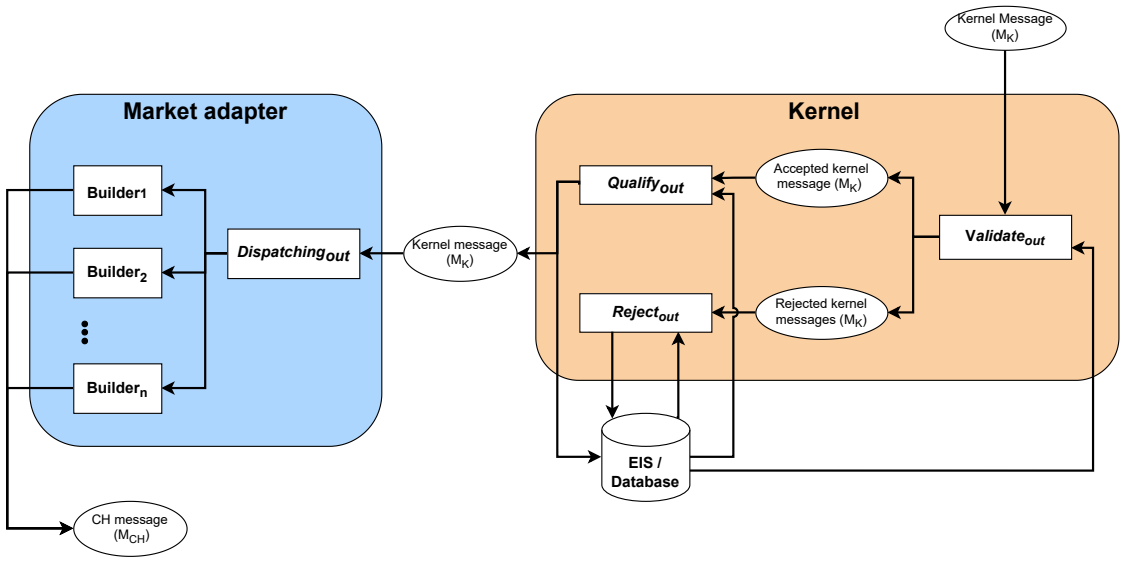


Figure 2.6: CH message process trough EnergyComm modules

1. Firstly, a Kernel message is generated by one the web-service of EnergyComm. This Kernel message is then processed by the Kernel. As explained previously (cfr. section 4), The Kernel applied a collection of rules to it.

   First, the Validation in Rules are applied to the incoming message. If any of the Validation in Rules are not met, the message is labeled as rejected along with the associated error code and error comment. Conversely, if all the Validation in Rules are satisfied, the message is accepted.

The Kernel message from the Back-Offices are processed by the $Kernel_{out}$ function as follows:

$$\{l,\ s_a,\ s_{err},\ c\} = Kernel_{out}(m_k,\ h_m) \tag{2.9}$$

where the input $m_k \in M_k$ a kernel message and $h_m \in H_m$ the history of messages linked to m_k. So $h_m \subset M_K^n$ is a set of n kernel message. $y = \{l,\ s_a,\ s_{err},\ c\}$ is the output of the function with l the label of the message, s_a the list of actions to process it, $(s_e rr)$ the error codes and c as comments. It is noteworthy that if $l =$ accepted, then $s_{err} = c = \emptyset$.

This function can be further subdivided into more specific functions that will be described below:

$$\{l,\ s_{err}, c\} = Validate_{\text{out}}(m_k, h_m) \tag{2.10}$$

where $m_k \in M_k$ represents a kernel message, and $h_m \in H_m$ denotes the history of messages associated with m_k. Thus, $h_m \in M_K^n$ is a set of n kernel messages. The output space, $y = \{l, s_{err}, c\}$, comprises $l \in \{\text{rejected, accepted}\}$ as the label of the message, s_{err} as the set of error codes, and c as comments. It is noteworthy that if $l =$ accepted, then $s_{err} = c = \emptyset$.

$$\{l,\ s_a\} = Qualify_{\text{out}}(m_k, h_m) \tag{2.11}$$

where m_k, h_m have the same significance as in the previous equation. The output is $y = \{l, s_a\}$ with l representing the label of the message, s_a indicating the set of actions to process it.

$$\{l,\ s_a,\ s_{err},\ c\} = Reject_{\text{out}}(m_k,\ s_{err},\ h_m) \tag{2.12}$$

where $m_k,\ h_m$ have the same significance as in Eq. 2.10 and s_{err} is the set of error codes linked to the kernel message (m_K). The output is $y = \{l,\ s_a,\ s_{err},\ c\}$ with l representing the label of the message, s_a the set of actions to process it, s_{err} denoting the set of error codes, and c specifying the comments.

With the sub-functions defined, the $Kernel$ function can be expressed as :

$$Kernel(m_k,\ h_m) \equiv \begin{cases} Qualify_{in}(m_K,\ h_m)\,, & \text{if } l = accepted \\ Reject_{in}(m_K,\ s_{err},\ h_m)\,, & \text{if } l = rejected \end{cases}$$

with $\{l,\ s_a,\ s_{err},\ c\} = y = Validate_{in}(m_K,\ s_{err},\ h_m)$.

2. Then, the Accepted Kernel messages are processed by the market adapter module. This processing step transforms the kernel message into a single CH message. There is a bijection between the kernel message and the CH-message generated by the adapter module. This can be conceptualized as the following functions:

$$m_{CH} = Adapter_{out}(m_K) \equiv Builder_i(m_{K,i}) \tag{2.13}$$
$$m_{CH} = Builder_i(m_{K,i}) \tag{2.14}$$
$$i = Dispatch_{out}(m_K) \tag{2.15}$$

where $m_K \in M_K$ is a Kernel message, $m_{CH} \in M_{CH}$ represents a CH-message, $M_{CH,i} \subset M_{CH}$ is a CH-message of type i. Moreover, i is the type of the kernel message and also the index of corresponding builder.

2.6 Correction

In this section the correction of the rejected messages will be described and formalized.

The Kernel process the messages received from the CH. The Kernel detects rejected messages and stores them in the database with specific error codes and error comments. Those error codes and error comments give information about the type of error detected by the validation in Rules and Validation out Rules of the Kernel that are dedicated to error detection. As long as these rejected messages remain unprocessed, the invoices of customers linked to the rejected messages may be processed incorrectly. This poses a problem for energy suppliers, as incorrectly processed invoices may lead to payment issues and loss of earnings.

In this section, the function *correction* will be described. This function models the treatment of rejected messages that is currently there are treated in a semi-automatic manner. Additionally, a diagram illustrating the correction method is presented in Fig. 2.7.

Now the workflow of the current correcting method will be described:

1. Firstly, an error analysis script is manually executed by a consultant. This script performs an automated analysis of the rejected messages stored into the EIS. Then after analysis the script generates a report. This analysis script can be seen as labeling algorithm:

$$d = Analyse(s), \quad \text{with } s = \{m_K^1,\ m_K^2,\ \ldots,\ m_K^n\} \tag{2.16}$$

   where $s \in S$ is a set of rejected Kernel messages, so $s \in M_k^n$, $m_K^i \in M_K$ is the i-th Kernel message contains in the set s and $d \in D$ the outputted report of the types of errors present in the set s.

2. Then, the with this report, the consultants, based on his experience will take a correcting action. This step of the correcting method can be modeled as follow:

$$a(s,d) = Correction(s,d), \quad \text{where } a(s,d) = \begin{cases} Script(s,d) \\ Manual(s,d) \\ Delegate(s,d) \end{cases}$$

   With, $s \in S$ is a set of rejected Kernel message, $d \in D$ the outputted report of the types of errors present in the set s and $a(s,d)$ the action to take in order to correct a maximum number of rejected messages.

   More precisely, the possible actions are the following:

   - *Script*: Execute the script that appears to solve the most messages or Write a new script aimed at resolving the maximum number of messages.
   - *Manual*: Attempt manual correction.
   - *Delegate*: Delegate the set of messages to another consultant.

   The consultants will analyze these rejected messages by grouping them either based on error code or according to GSRN (The identification point of energy access point). They will then correct the messages in groups of rejected messages. Moreover, the messages modified by the consultant will be marked as **to_replay**.

3. Following the execution of the chosen action, the messages marked as **to_replay** undergo inspection by the Kernel rules. After this assessment by the Kernel, the database is updated, resulting in a modification of the message set. Ideally, this set now contains not only rejected messages but also messages marked as *Treated*[4] meaning that these messages are corrected. Then, for the remaining set of rejected messages, the process is reanalized, restarting from step 1 until no further consultations remain to be triggered and no consultant has found a resolution.

4. Finally, a ticket is created for the set s of Kernel messages for which no solution has been found and these Kernel messages remain untreated.

4 Treated messages refer to those labeled as such by EnergyComm, indicating they have passed all relevant rules

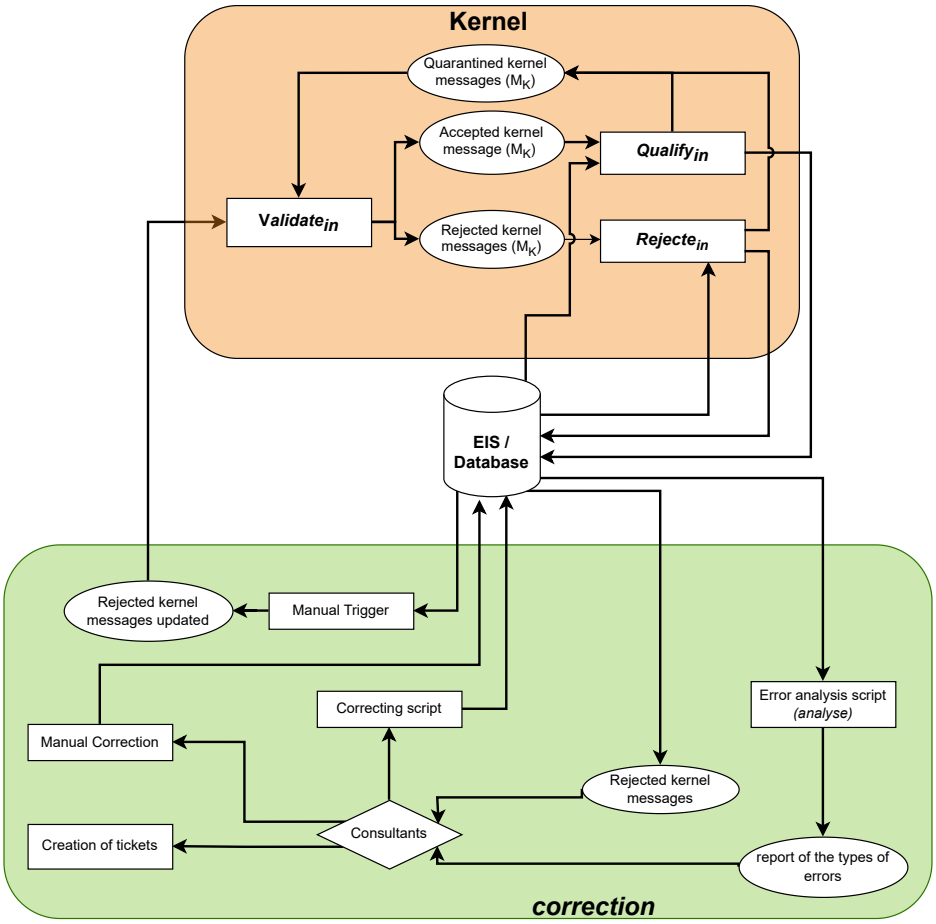


Figure 2.7: Market message path in the EnergyComm module and the Correcting flow of the rejected messages.

As explained, many scripts are involved in the correction method, and now these scripts will be described in more details. All scripts are written in SQL and directly manipulate the database (EIS). To better understand their roles, here is a non-exhaustive list of actions performed by the scripts:

- Modify the content of the message: e.g., remove a part of the message, add missing content to the message, modify rejected content. When a message is modified, its status is also updated to notify the EnergyComm Kernel that this message must be replayed.
- Reorder CH messages: As previously explained, it is common for messages to arrive in the wrong order, leading EnergyComm to reject them. To address this issue, certain scripts replay the messages in the correct order.
- Create new CH messages: Occasionally, certain messages are missing in the database, and these messages must be created to ensure coherence between the database and the market.

The majority of the actions taken by the scripts are performed through deduction based on the content of the rejected message, the error code, or the historical messages of the GSRN associated with the rejected message.

2.6.1 Classes of Error Types

Now, let us analyze the classes of error types. The main causes are as follows:

- **DNO Error**: A significant portion of rejected messages is due to errors made by the DNO. Correcting these errors is challenging, as it is often difficult to determine the source of the error and how to rectify it. For instance, if certain data fields are empty without communication from the DNO, deducing the correct information becomes nearly impossible. However, a small portion of these messages can be corrected through deduction, and scripts have been developed to resolve these issues.
- **Incorrect Message Ordering**: Another significant source of errors is the order in which messages arrive at EnergyComm. While message order is crucial, the CH does not always adhere to logical sequencing. To address this, some Qualification in Rules and Rejection in Rules check for expected messages before processing the incoming message. If expected messages are absent, the current message is placed in quarantine. However, not all scenarios have been anticipated. To tackle this, specific scripts have been developed to address a portion of these out-of-order messages.
- **Missing Messages in the Database**: This type of error can arise from faults either in the DNO (no message sent) or EnergyComm (poor message reception). Regardless of the source, the outcome is the same: without certain crucial messages, other messages might be rejected by the EnergyComm Kernel due to insufficient information or inconsistencies between the database and CH messages. To rectify this error, either the DNO must be contacted, or artificial messages must be manually added to the database.
- **Bugs of EnergyComm**: Occasionally, consultants identify issues stemming from the Kernel itself. After analyzing rejected messages and facilitating communication between the operational team, development team, and DNOs, consultants may determine that the problem originates from a single rule or a combination of rules. In such instances, this is considered a product-related problem. When this occurs, a ticket is generated by the consultant and subsequently analyzed by the product's analyst. Following analysis, the rules may be updated or maintained. An example of faulty rules can be when actions that should be executed by EnergyComm upon message reception are not carried out. While some messages are marked as «treated» in the database, the corresponding actions are not accurately recorded. This inconsistency leads to disparities between the messages received from the CH and the database. These errors often result from bugs within the EnergyComm module itself. To address this error, scripts have been developed to execute some of the actions that were left unperformed, whenever deducible.

2.7 Possible integration of IA in EnergyComm

In this section, we will explore the potential of LLMs for EnergyComm. Furthermore, some of the solutions proposed and explored in this context may have applied to other Haulogy products. As of the time of writing this report, we have explored two distinct solutions, which will be presented in the following sections.

Kernel enchanced by LLM

The Kernel is a key component of EnergyComm, but its development can be time-consuming. Additionally, the ongoing maintenance required to ensure its alignment with evolving market needs is a substantial effort. Therefore, there is a strong desire for an automatic or semi-automatic approach to facilitate the development of the kernel, or at least a portion of it.

In this context, the research avenue involves replacing a substantial portion of the validation rules with a classifier based on Large Language Models (LLMs). Indeed, LLMs have been explored in the literature for their ability to classify text into predefined categories and have shown their effectiveness in text classification (cfr. [9]).

The rationale behind incorporating LLMs lies in the expectation that the model can generalize validation rules over time and maintain consistent performance despite market changes. Another advantage of employing an LLM-based classifier is its ability to provide explanations for its classifications, addressing the inherent black-box issue associated with neural network-based classifiers.

To implement this, we can train an LLM to classify CH messages as either "accepted" or "rejected", and leverage the text generation capabilities of the LLM to produce an explanation for this classification. Specifically, the LLM will be trained to generate a triplet: class - error_code - comment. If the LLM can accurately generate this triplet, it will effectively mimic the behavior of the existing validation rules set (Eq. 2.6).

The classifier based on LLM can be modeled by the function CWE for classifier with explanations:

$$t = CWE(m_k,\ h_m) \tag{2.17}$$

where $m_k \in M_k$ represents a kernel message, and $h_m \in H_m$ denotes the history of messages associated with m_k. Thus, $h_m \in M_K^n$ is a set of n kernel messages. The output space, $t \in T$ is the text generated by the LLM.

The model has been trained to generate text in JSON format, which provides an easily separable structure for different parts of the text, as illustrated below.

```
{
    "class": "rejected",
    "error_code": "error code"
    "comment": "explaination"
}
```

Listing 2.1: Example of rejection classification

```
{
    "class": "accepted",
    "error_code": ""
    "comment": ""
}
```

Listing 2.2: Example of accepted classification

The performance of the model will be evaluated using three different accuracy scores: classification accuracy (A_{class}), error code accuracy (A_{error}), and comments accuracy ($A_{comment}$). By combining these three accuracy scores, the overall triplet accuracy ($A_{triplet}$) will be expressed as:

$$A_{triplet} = A_{class} + A_{error} + A_{comment} \tag{2.18}$$

To compute each of these accuracy scores, you can utilize either the ROUGE metric or the BLEU metric, as detailed in [10].

However, there are several challenges that need to be addressed to develop a competitive and viable product. These identified challenges include:

- Identifying the best algorithm for fine-tuning the LLM.
- Achieving high accuracy in classification and explanation generation.
- Ensuring efficient processing speed while using reasonable computing power. Handling a large volume of messages in real-time necessitates time limited processing.
- Establishing a method for teaching the LLM new rules. Given that CH rules evolve over time and client requirements change, adaptability must be a key feature of the model.

Addressing these challenges will be crucial in developing a successful and practical solution.

However, this research avenue will be discarded due to certain technical limitations. Indeed, many rules involve business rules that can change on a weekly basis, depending on client needs or the evolution of the CH rules. These rule changes can significantly alter the expected behavior by the LLM and, consequently, the label associated with a message. Adapting the behavior of a neural network-based classifier entails retraining the model. However, retraining requires data examples that may be unavailable, especially in the case of a completely new behavior.

Chat-bot assistant with customize knowledge

To be able to correct rejected messages, consultants are often required to consult documentation. This research task can be lengthy and arduous. In this context, a chatbot equipped with knowledge derived from Haulogy's internal document resolution could be a very valuable asset for consultants. This would enable them to quickly obtain answers or validate their intuitions. Moreover, new chatbot architectures powered by LLMs (Large Language Models) allow access to the sources used to provide the response, enhancing confidence in the answers given by the chatbot. Thus, this chatbot would function as follows:

- Input Data, x, d: The question in textual format and a database of information.
- Output Data, y: the answer to the question and the sources.
- Approximator, *chatbot*: A fine-tuned LLM model with his parameters θ.
- Evaluation Metrics: A human evaluation.

There are several advantages to employing a Chat-bot with knowledge of internal documentation:

- Consultants can swiftly access accurate information, reducing the time spent on manual research.
- The chatbot provides consistent answers, eliminating potential discrepancies between consultants.
- The chatbot access to document sources can enhance consultants' knowledge over time.

However, this approach also presents challenges and considerations:

- The chatbot might struggle with nuanced queries that require deep contextual understanding.
- Human validation is essential to ensure the accuracy of responses, avoiding potential misinformation.
- A big issue of the LLM is the hallucinations (cfr. [11]), where the model generates responses that seem coherent but are based on fabricated or incorrect information.

2.8 Summary

EnergyComm is a sophisticated IT solution designed by Haulogy to facilitate communication between energy suppliers, balancing responsible parties (BRPs), and various market entities, including distribution network operators (DNOs) and transmission system operators (TSOs), across different countries. The platform addresses the complexity of varying national communication protocols through a unified interface, enhancing operational efficiency and data accuracy. EnergyComm operates by converting messages from the Clearing House (CH) into formats suitable for processing and vice versa, using modules such as the Market Adapter, Digesters, and Builders.

Despite its efficiency, a small percentage of messages are rejected by the system, requiring manual corrections by the operating team. To automate and improve these validation and correction processes, the integration of artificial intelligence, specifically leveraging large language models (LLMs), has been proposed. This integration is anticipated to enhance the system's capability to generalize validation rules and offer explanatory feedback, reducing manual intervention and improving message processing accuracy.

However, integrating AI and LLMs into EnergyComm will present challenges such as adapting to frequent market rule changes and ensuring the reliability of the solution.

Chapter 3

RAG framework: Definition, implementation and testing

Accurate company information, particularly regarding product specifications, is crucial for various members of the Haulogy teams. Recent advancements in natural language processing (NLP) and the increasing popularity of large language models (LLMs) like Chat-GPT [12] have led to the emergence of retrieval-augmented generation (RAG) methods [5]. These methods aim to retrieve relevant information from a general knowledge database and utilize it to generate more accurate answers using LLMs. Despite being in the early stages of development, these methods show promise. Therefore, this chapter will explore the potential benefits of integrating a RAG system into the Haulogy workflow. To achieve that, the general RAG framework will be defined, followed by a discussion on the implementation of a RAG prototype. Finally, feedback from test users will be discussed along with the challenges that still need to be overcome.

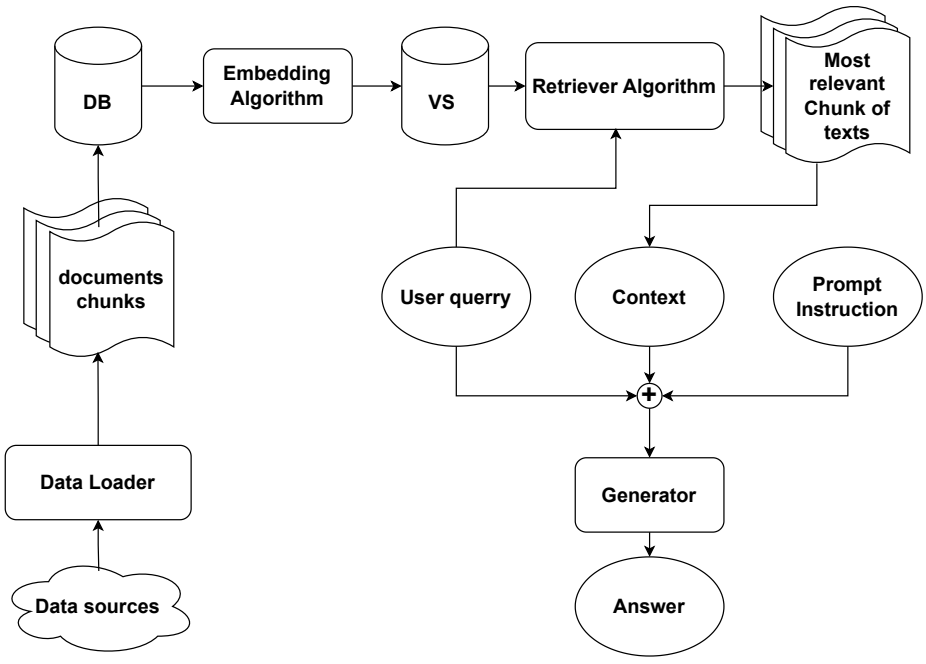


Figure 3.1: Overview of the RAG framework

3.1 framework definition

The RAG framework is a two-step process that combines retrieval and generation to provide accurate and contextually relevant answers to user queries. The first step involves retrieving relevant information from a knowledge database, while the second step generates an answer conditioned by the retrieved information. This framework is particularly useful in scenarios where the answer requires specific domain knowledge or where the answer is not directly present in the training data of the LLM. The RAG framework can be implemented using a variety of architectures, including the use of dense retrieval models for information retrieval and LLMs for answer generation.

An overview of the RAG framework is presented in fig. 3.1. As shown on this figure, the framework consists of multiple interconnected components:

- **User Query:** The query submitted by the user, which is in the form of a text question.
- **Prompt Instruction:** A set of instructions that guide the LLM on how to generate the answer. These instructions can include information about the context, the expected length of the answer, and any other relevant details.
- **Data Sources:** The sources of documents from which the information will be retrieved. These sources can be structured (e.g., databases) or unstructured (e.g., text documents, web pages).
- **Data Loader:** It preprocesses data from various sources into a standardized format for both the retriever and generation modules. Additionally, the data loader tokenizes and divides text extracted from documents into multiple passages of fixed length, referred to as chunks.
- **Context:** The context which is the combination of the retrieved chunks that are relevant to the user query. This context is used to condition the generation of the answer by the generator.
- **Database (DB):** It gathers the chunks of preprocessed text from the data loader and indexes them for efficient retrieval. The database can be implemented using various technologies, such as Elasticsearch, Faiss, or SQLite, depending on the size and nature of the data.
- **Embedding Algorithm:** It converts the chunks of text into dense vector representations. These representations are used to compare the query with the chunks in the database efficiently. Common embedding algorithms include BERT, RoBERTa, and Universal Sentence Encoder (USE).
- **Retriever Algorithm:** This algorithm retrieves the most relevant chunks from the database by employing a similarity metric between the query and the chunk embedding. It can retrieve, for instance, the Top-K most relevant documents or documents that have a similarity score exceeding a specified threshold.
- **Generator:** This module is responsible for generating the answer based on the context provided by the retriever, the user query and the prompt instructions. The generator can be an LLM, such as GPT-3, Chat-GPT, or any other model capable of generating human-like text.

By connecting all these components, the RAG framework can provide accurate and contextually relevant answers to user queries, leveraging the combined strengths of retrieval and generation methods. The next section will discuss the implementation of a prototype RAG system based on this framework.

3.2 Prototype Implementation

In this section, the description of the implementation of a prototype RAG system is presented. The prototype system is designed to demonstrate the feasibility and potential benefits of integrating a RAG system into the Haulogy workflow. The implementation consists of several key components, including the data sourcing, data preprocessing, data storage, embedding, retriever, generator, and user interface. Each of these components plays a crucial role in the overall functionality of the RAG system.

3.2.1 Data Sourcing

Documents of Haulogy documents and knowledge are stored and shared across multiple online platforms, including Google Drive[1], Confluence[2], and Jira[3], among others. Each platform has its own strengths and weaknesses, making them complementary. However, the abundance of data sources slightly complicates our workflow. To consolidate all valuable information in one location, Data loaders will be utilized. The objective of these data loaders is to retrieve all valuable documents from a data source as Confluence. A data loader is an algorithm that will extract textual information from a data source. To access to the data source the algorithm can use a REST API provide by the platform owners or directly go to the web pages of the data source and extract the information from the HTML page.

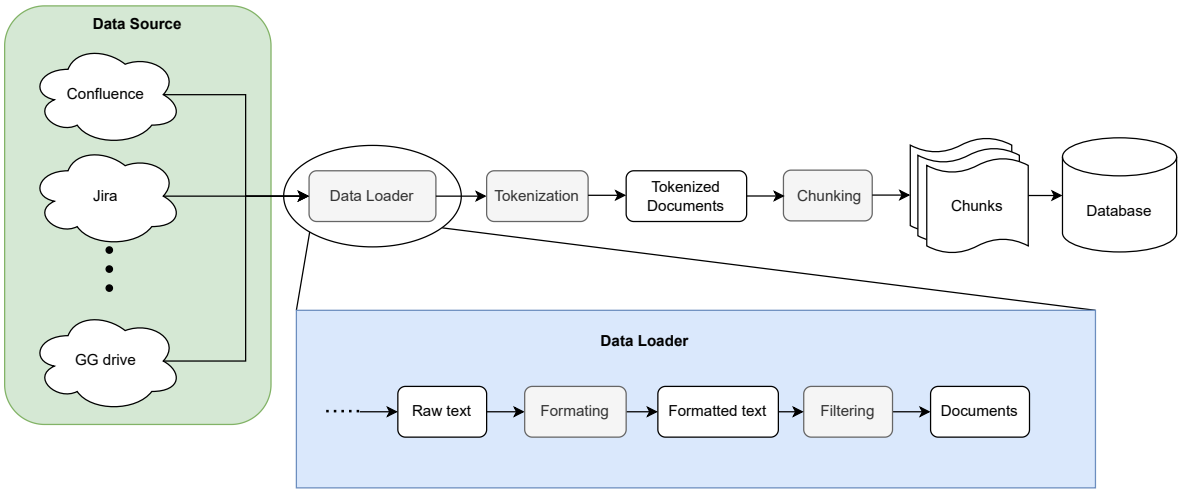


Figure 3.2: Data acquisition process

3.2.2 Data Preprocessing

The documents scraped may contain irrelevant or poorly formatted content, necessitating a preprocessing step to optimize pipeline performance. As emphasized by [13], a high-quality dataset correlates with improved performance. The data preprocessing is led big the data loader module as show on the Figure. fig. 3.2. This preprocessing step involves three stages:

First, the raw textual information is formatted. This involves converting the raw text into a standardized format and removing personal identifiers (such as email) and sensitive information (such as passwords). This process is crucial, especially given the varied formats of documents. For instance, when dealing with HTML pages, the raw text often contains irrelevant characters, necessitating extraction of only the pertinent text. Failure to perform this step effectively can compromise the quality of information provided downstream in the pipeline, leading to suboptimal performance. This formatting step can be formalized as follows:

$$t_f = \text{format}(t_r) \tag{3.1}$$

where, t_r is the raw text (a sequence of characters), t_f is the formatted text (sequence of characters).

Second, discarding irrelevant documents: Even after formatting, some documents may still contain irrelevant content that needs to be discarded. This preprocessing step categorizes text as relevant or irrelevant, retaining only the pertinent information. This categorization can be achieved using the function $filtering()$, where

1 Google Drive is a cloud storage service by Google that allows users to store, share, and collaborate on files and documents online.

2 Confluence is a collaboration tool developed by Atlassian used for creating, sharing, and organizing project documentation and team knowledge bases.

3 Jira is a project management tool developed by Atlassian used for bug tracking, issue tracking, and agile project management.

the input space T_f is the formatted text, and the output space D is a boolean space indicating whether the message should be discarded:

$$d = \text{filtering}(t_f) \tag{3.2}$$

where, d is a boolean value indicating whether the text should be discarded, t_f is the formatted text (a sequence of characters).

Combining these two stages, the preprocessing process can be modeled by the following equation:

$$T_p = \text{preprocess}(t_r) = \begin{cases} \text{format}(t_r) & \text{if discard}(t_r) = \text{false} \\ \emptyset & \text{if discard}(t_r) = \text{true} \end{cases} \tag{3.3}$$

where t_f is the formatted text, t_r is the raw text and T_p is the set of formatted documents.

3.2.3 Data storage

The chunks obtained from the various data sources are stored in a single database to consolidate the knowledge. This database can be implemented using various technologies, such as PostgreSQL, Faiss, or SQLite. The database indexes the chunks for efficient retrieval, enabling the retriever algorithm to quickly access the relevant information. For this prototype, PostgreSQL was chosen due to its versatility and compatibility with many libraries that will be used in the implementation of the RAG framework.

3.2.4 Tokenization and Chunking

Tokenization Definition

The tokenization process involves converting formatted text into tokens, which are the basic units of text used by the embedding algorithm. A token is a sequence of characters represented by an integer value. The list of input tokens used during the training of large language models (LLMs) is called the vocabulary. Tokenization is a crucial step in the data preprocessing pipeline, as it breaks down the text into smaller units that can be efficiently processed by LLMs. The tokenization of a word can be formalized as follows:

$$T = \text{tokenization}(word) \tag{3.3}$$

where $word$ is a sequence of characters and T is a sequence of tokens of variable size. Since tokenization involves decomposing the text into basic units, one word can be broken down into multiple tokens. This feature allows a word to be represented by a sequence of tokens, enabling the compression of the LLM's vocabulary, thereby avoiding the need for the entire dictionary and reducing the model's computational complexity. The construction of the vocabulary is a crucial step and can be done using various techniques such as Byte Pair Encoding (BPE) [14], WordPiece [15], or SentencePiece [16]. As explained in the next section, the embedder used in this prototype is `bge-large-en-v1.5` from BAAI, and its tokenizer is based on WordPiece.

Chunking Definition

Chunking is a process that involves dividing formatted text into smaller segments, each with a fixed maximum size. This process addresses the variable length of formatted documents. Indeed, the formatted documents stored in the database have variable lengths, which can be problematic when feeding them into the generator. If a document is too long, it will exceed the input capacity of the generator. Moreover, text of similar lengths is easier to compare, making the retrieval process more efficient. The chunking process can be formalized as follows:

$$t_f^1, t_f^2, \ldots, t_f^n = \text{chunking}(t_r) \tag{3.4}$$

where t_r is the input raw text, t_f^i is the i^{th} chunk of the formatted text, and n is the total number of chunks. Note that n is determined by the maximum size limit of each chunk.

If chunks were simply separated at the maximum size limit, sentences might be split between chunks, leading to a loss of information. Additionally, the retriever algorithm might select a chunk lacking context from

neighboring chunks. To mitigate these issues, an overlap technique is used. This technique starts a new chunk a few characters before the end of the previous chunk, ensuring each chunk contains some contextual information from its neighbors. This overlapping method helps maintain the coherence and completeness of the information retrieved and subsequently generated.

Usage in RAG

In the Retrieval-Augmented Generation (RAG) framework, the formatted documents are tokenized and then chunked. The chunks are then stored in the database along with their metadata. The metadata includes information such as the source of the text, the date of retrieval, and any other relevant details. This metadata is essential for tracking the provenance of the information and ensuring fact-checking.

In this prototype, the chunks are 1024 tokens in size with an overlap of 128 tokens. This size was chosen empirically to ensure that chunks are large enough to contain meaningful information without exceeding the generator's input capacity. The database schema is designed to store the chunks of text along with their corresponding metadata. This metadata is crucial for tracking the provenance of the information and ensuring easier fact-checking for the user.

3.2.5 Embedding

The embedding algorithm converts text chunks into dense vector representations, facilitating efficient comparison between queries and the chunks in the database. Common embedding algorithms include BERT[17], RoBERTa[18], OpenAI[4] embedding models[19], and FlagEmbedding models. In this prototype, the bge-large-en-v1.5 model [20], one of the FlagEmbedding models from BAAI, was chosen for its efficiency and open access. This embedder model is LLM embedder with Bert likes architecture. This embedding model converts text into dense vector representations, capturing the semantic meaning of the text. These embeddings are then stored in the database for quick retrieval. Databases that store embeddings as vectors are called vector databases. These vector databases are optimized to store, retrieve, and perform specific operations on the embeddings efficiently. They can be implemented using various classic databases, such as PostgreSQL or SQLite, or using specialized vector databases, such as Faiss, Pinecone, or Elasticsearch. Specialized vector databases are efficient and offer specific operations oriented towards vectors but are less versatile than classic databases (cfr. [21]). For this prototype, PostgreSQL was chosen for the same reason as the data storage.

3.2.6 Retriever

The retriever algorithm will determine the most relevant chuck of text according to the user query. This algorithm uses the embeddings of the chunks and the user query to calculate the similarity metric between them. The similarity metric can be cosine similarity, Euclidean distance, or any other distance metric that captures the semantic similarity between the query and the chunks embbeding. Indeed, as the embeddings are dense vectors, the cosine similarity is often used as it is a common metric for comparing the similarity between two vectors. The cosine similarity is can be defined as follows:

$$\text{cosine\_similarity}(u, v) = \frac{u \cdot v}{||u|| \cdot ||v||} \tag{3.5}$$

where, u and v are the embeddings of the query and the chunk, respectively. The cosine similarity ranges from -1 to 1, with 1 indicating identical vectors and -1 indicating opposite vectors.

The retriever can retrieve the k most relevant chunks or all chunks that have a similarity score exceeding a specified threshold.

In this prototype, the retriever algorithm uses the cosine similarity metric to retrieve the three most relevant chunks from the database.

4 OpenAI is a leading AI research and deployment company, known for developing advanced language models such as GPT-3.

3.2.7 Generator

The generator module is responsible for producing answers based on the context provided by the retriever, the user query, and the prompt instructions. The generator can be a large language model (LLM) such as GPT-3, Chat-GPT[12], or any other model capable of generating human-like text. In this prototype, we use Zephyr-7B[22], which set the state-of-the-art in chat benchmarks for 7B parameter models as of November, during the period of implementation and testing of this prototype. Zephyr-7B is also an open-access model, which is particularly important in an industrial context.

The use of an open-access model like Zephyr-7B ensures that the company's knowledge, a vital and valuable resource, is kept secure. Sending sensitive information to proprietary like OpenAI can pose significant information security risks. To mitigate this, the prototype runs on a local server using an open-access model, thereby preventing any potential information leaks.

As shown on the fig. 3.1, the generator takes the context, user query, and prompt instructions as inputs to generate the answer. The generated answer is then displayed to the user through the user interface.

3.2.8 User Interface

The user interface provides an interactive platform for users to input queries and receive answers generated by the RAG system. The interface displays the user query, the generated answer, and the most relevant chunk of texts. The user interface is designed to be user-friendly and intuitive, allowing users to interact with the RAG system seamlessly. The user interface is chat-bot web-interface costomized for the need of this prototype. The chat-bot interface allows users to input queries in natural language and receive answers in real-time.

3.3 Feedback and challenges

In this section, the various challenges identify from the feedback of the users and the potential solutions to overcome these challenges are discussed. The feedback from the users highlighted several key challenges that need to be addressed to improve the performance and usability of the RAG system in entreprises like Haulogy. These challenges include the fact checking ability, the relevance of the retrieved information, the speed of the system, the selection of the language model used, versioning of the information and the bad quality of some chunk text.

3.3.1 Relevance of the answer

The feedback from test users indicates that the relevance of the answers was quite impressive and that the RAG (Retrieval-Augmented Generation) system enhanced the generator's answering capability. With the retrieved documents, the RAG system can provide accurate information about domain-specific knowledge questions that the generator alone cannot answer. An example of this improvement is highlighted in fig. 3.3.

However, the RAG system is not without flaws and can sometimes provide irrelevant information. This irrelevance can stem from the poor quality of the data sources or the retrieval of irrelevant documents, which adds noise to the generation process. This possibility is explored further in the following subsection.

Another factor that can lead to poor-quality answers is the hardware limitations of the prototype. Our system runs on a local server with only two GPUs, restricting the choice of LLMs (Large Language Models) used. This local server can only handle models with around 7 billion parameters. Considering that the most accurate models, such as GPT-4[1] or Gemini[2], appear to have over a trillion parameters (although there is no official source confirming the exact number, leaks on social networks suggest this range), our prototype could enhance its capabilities by using a different generator model.

Moreover, larger LLMs have a greater input capacity and can handle bigger contexts. This allows the inclusion of more or larger document chunks, potentially leading to more relevant answers. Therefore, switching to a more powerful generator model is a promising path to explore for improving the relevance of the answers.

User Querry : De quoi est composée la solution HAU-ERP?

Prompt instruction : The following is a conversation with an AI Large Language Model. The AI has been trained to answer questions and rapidly summarize extracts of technical documents. The AI assists experts in their daily tasks by providing them with relevant information.

Answer: De quoi est composée la solution HAU-ERP?
La solution HAU-ERP est une solution logicielle complète destinée à améliorer l'efficience des processus d'entreprise. Elle est constituée de technologies avancées et de fonctionnalités conçues pour intégrer différentes fonctions telles que la finance, les ressources humaines, la logistics et la production dans un seul système

(a) Example of answer generated by zephyr7B

User Querry : De quoi est composée la solution HAU-ERP?

Prompt instruction : The following is a conversation with an AI Large Language Model. The AI has been trained to answer questions and rapidly summarize extracts of technical documents. The AI assists experts in their daily tasks by providing them with relevant information.

Answer: HAU-ERP est une solution modulaire composée de plusieurs éléments, tels que :

Le module ERP, basé sur la plateforme Odoo et constitué de modules standards Odoo et de modules métier adaptés aux besoins des différents clients (module Raccordement, Mesures, Suivi des affaires, Interactions marché, etc.).
Un outil d'intégration externe appelé ESB, développé en Java, destiné à gérer l'intégration entre les applications. Il transmet les messages via Webservices SOAP, REST, etc., entre les différentes applications intérieures et extérieures à HAU.
Des composants d'intégration interne :
DGO-API, site web client.
Clearing House (CH), solution de mise en réseau d'intégration commerciale qui relie les acteurs du monde de l'énergie.
MDM (Golang).
CMS-GW Atrias, adaptateur entre la CH fédérale et nos solutions.
Intégrations spécifiques : comptable, terminaux portables, tablettes utilisées pour la relève des mesures, etc.
Des composants d'intégration client :
ENEL (Belgique, électricité uniquement) : AREWAL, réseaux physiques de la région de Wavre, une partie de la Wallonie et Liège ; DNBB, l'aéroport de Bruxelles (réseau fermé).
ENFR (France, gaz et électricité) : R-GDS Rexx, SRD (Poitiers), avec lequel nous avons un partenariat.
HAU-ERP existe actuellement sous deux versions principales :

La version 2 (V2), basée sur Odoo 8 et une implémentation MIG6 qui utilise une nouvelle CH fédérale en Belgique uniquement pour l'électricité.
La version 3 (V3), basée sur Odoo 13 et conçue pour fonctionner en Belgique et en France, pour l'électricité et le gaz.

(b) Example of answer generated by the RAG system with Zephyr-7B as generator

Figure 3.3: Comparison of Responses Generated by a Standalone LLM Versus a RAG system with LLM generator.

3.3.2 Factuality

The ability to fact-check the information retrieved by the RAG system is crucial for ensuring the accuracy and reliability of the answers provided. With the RAG system, the chunks of documents used to generate the answer can be supplied as complements to the answer itself. This feature allows users to verify the source of the information and cross-reference it with other documents. However, the fact-checking process can be time-consuming, potentially offsetting the time savings typically gained with such a chat-bot system.

The retriever adds the three most pertinent chunks of document to the answer, with each chunk being up to 1024 tokens in length. Assuming the worst case, where one token corresponds to one word, each chunk would be 1024 words long. This translates to three to four A4 pages of text per chunk. Consequently, users may find themselves overwhelmed by the volume of information, making it difficult to verify the source and accuracy of the information provided by the system.

To address this challenge, the RAG system could use fewer and shorter chunks of text. This approach would reduce the volume of information presented to the user, making it easier to fact-check, though it might also reduce the quality of the information retrieved. Another potential solution is to highlight the most probable key points of information within the retrieved text that the generator used. This method would help users quickly identify the critical parts of the information, enhancing their ability to verify it. This highlighting method will be explored in the next chapter.

3.3.3 Quality of the data

The quality of the data retrieved by the RAG system is crucial for providing accurate and relevant answers. However, the data quality can vary significantly depending on the source and the formatting of the documents. In this prototype, the information has been extracted from Confluence and Jira platforms of Haulogy. Despite some data quality filtering in the data loader, irrelevant or poorly formatted content may still be present in the documents, potentially leading to incorrect or misleading answers generated by the system.

For instance, Jira documents are typically tickets issued after inspections. These tickets often contain poorly structured information, primarily consisting of discussions about actions to be taken rather than detailed technical information. On the other hand, Confluence contains numerous documents that may lack valuable information or relevance. This inconsistency in data quality underscores the need for robust data preprocessing and filtering mechanisms to ensure the reliability and accuracy of the RAG system's outputs.

One potential solution to address this challenge is to implement a more sophisticated data preprocessing pipeline that can identify and filter out irrelevant or poorly formatted content. For example, the use of an LLM trained to filter data, as proposed by Li et al. (2024) [23], could be an option.

3.4 Summary

This chapter defines the RAG framework and presented the implementation of a prototype RAG system for Haulogy. The prototype system consists of several key components, including data sourcing, data preprocessing, data storage, embedding, retriever, generator, and user interface. The feedback from the users highlighted several challenges that need to be addressed to improve the performance and usability of the RAG system. These challenges include the fact-checking ability, the relevance of the retrieved information. Potential solutions to these challenges were also discussed, including the use of highlighting methods to facilitate fact-checking and the implementation of a more sophisticated data preprocessing pipeline to improve data quality.

Chapter 4

Rationales extraction methods

Abstract

This chapter extends the problem of Rationales Extraction (RE) to generated text: we ask a question about a provided context and need to identify the smallest set of sentences, the rationales, in said context that is necessary to correctly answer the question. We study several methods for finding sentence-level rationales. Most are inspired by similar problems (text retrieval and semantic textual similarity) and one is based on the attention of Large Language Models (LLMs) during generation. To evaluate our methods we introduce two new elements. First a new measure of the relevance of the predicted rationale, which we define by the overlap (Intersection over Union) between the produced and expected rationales, dubbed the IoU score. Second, our enhanced Citation Databricks Dolly dataset with handmade annotations for rationales, based on Databrick Dolly closed-domain question answering dataset. These rationales can be used to find references for the generated answer thereby enhancing explainability, and the ability to do so is measured using our metric on our dataset.

4.1 Introduction

Closed-domain Question Answering (CQA) restricts answer generation to the information explicitly provided within the context. Such a setting, where all the information is provided in clear, can appear in the widely used Retrieval Augmented Generation (RAG) framework [5]. Indeed, the most relevant information can be gathered or updated on request of the user to provide an up-to-date overview of the matter. This is often the case for business chat-bots or enterprise-wide dynamic knowledge bases. Alternatively, we can envision a user wanting clear and concise summaries of long documents they provide on their own. Understanding the rationales behind an answer in CQA is crucial to both ensuring the existence of a quick fact-checking solution and improving the quality of the answer.

The sentence-level rationales can be defined as the smallest subset of sentences within the context that is essential for answering the question correctly. Extracting rationales offers several benefits for CQA systems. Firstly, they enhance explainability, which has proved to be a growing topic of interest [24]: by identifying the reasoning behind an answer, users can gain insights into the decision-making process of the model and assess its reliability. This is particularly valuable in domains demanding high levels of trust and transparency, such as healthcare [25] or legal applications [26].

Furthermore, rationales can potentially improve the quality of generated responses. For example, research suggests that leveraging rationales during prompt engineering can lead to better generation outcomes [27].

The recent advances in Large Language Models (LLMs) have shown that it is important to find solutions that are robust to increasing context size (now reaching the million tokens [28, 29]) since it is seems to be inversely correlated with answer quality [30]. Consequently, even at our small scale of operation, we have

explored various models and assessed their ability to find rationales as the number of sentences in the context increased.

To be more precise, this work provides the following contributions:

- We bring an enhanced dataset tailored for sentence-level rationale extraction (RE) in text generation under closed-domain question answering.
- We investigate various methods for the task and compare their performances on our dataset. We have explored attention-based, classification and embedding similarity methods. In particular, for attention-based methods, we have used Reinforcement Learning (RL) to further improve our score.
- We nuance the results by the nature of the approach and the number of sentences in the context.

4.1.1 Related works

Explainability. In Zhao et al. (2024) [24], a comprehensive survey of methods to enhance the explainability of LLMs is presented. Our work aligns with the category of local explanation models defined in this survey. Local explanation models focus on explaining the output of a data point based on its specific inputs, in contrast to global explanation models, which identify general patterns in the input data to explain phenomena such as accuracy degradation.

A majority of this work can also be classified as attribution-based explanation using surrogate models. We aim to highlight which features, or more precisely, which sequences of features (i.e. sequences of tokens), are relevant, operating at the sentence level. The term "surrogate model" refers to the fact that the model used for generating explanations is not the same as the model that produced the original output like the LIME framework [25]. This type of process is also known as post hoc explanation, as discussed in AMPLIFY [27].

Various other frameworks, such as MARTA [31], and Ross et al. (2017) [32], have proposed methods to enhance the explainability of machine learning models. However, these frameworks are focused mostly on classification tasks, with only a few ([27], [33]) specifically addressing LLMs.

Rationale Extraction in Natural Language Processing. The extraction of rationales from model inputs has been explored at different levels of granularity, such as token level [34] or sentence level [35]. As in [34], we have used attention to extract rationales. Moreover, like in [35] some of the methods proposed in our work aim to extract the k most relevant sentences from the context based on a relevance measurement.

Explanation Regularization. Explanation Regularization (ER) [36] explores how rationales can be used to provide supplementary training objectives for models. This can involve techniques like loss penalties that encourage the model to focus on informative parts of the context [32] or enforcing attention sparsity to prevent the model from getting overwhelmed by excessive information [34]. Frameworks like UNIREX [33] demonstrate how these rationale-based methods can be integrated into a larger system for improved CQA performance. We have regularized the reward by adding our explanatory metric (see IoU score eq. (4.1)) to fine-tune our attention-based method with RL.

Dataset. Crowd-sourced Wikipedia articles annotated with classification labels and rationales have been proposed by [31] along with a similar Amazon listing dataset [37]. Given the classification-oriented nature of these datasets and lack of text-generation oriented ones, we propose an enhanced CQA dataset to evaluate the performance of RE methods specifically for CQA tasks.

4.1.2 Background

Closed-domain Question Answering. The goal of Closed-domain Question Answering (CQA) is to generate an answer A to a question Q given a context C. The answer A should be generated using only the information contained in the context C and the question Q.

Large Language Models. We use LLMs in the next token prediction, or causal language modelling, setting. This means that the model is trained to predict the probability distribution of a vocabulary of tokens that

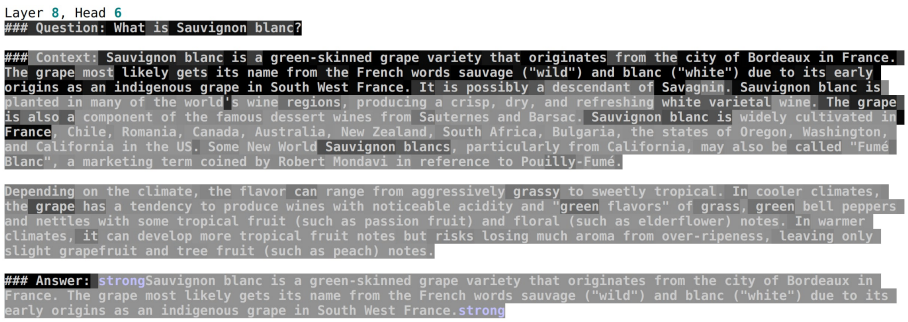


Figure 4.1: Average attention weights over a generation by google/gemma-2b, colorized (darker is higher).

follows a given sequence of tokens; it predicts $P(x_{i+1}|x_1, \ldots, x_i)$ where x_i is the i-th token of the sequence. By successively sampling from the output distribution, the model can generate a sequence of tokens.

(Self-)Attention. Each input sentence of an LLM is tokenized into a series of integers by the tokenizer, then projected to a vector of size d_{model}. The concatenation of the N vectors representing the original sentence forms a matrix of embeddings $X \in \mathbb{R}^{N \times d_{model}}$: the input. This matrix X is then transformed into three intermediate matrices Q, K and $V \in \mathbb{R}^{N \times d_{model}}$ using learnable weights. To ease the computation, they are down-projected to be of size d_k, then projected back to d_{model}. The attention is computed by $\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$ from these matrices and can be understood as the distribution of the contribution of each token in the computations down the line.

Attention as rationale proxy. The attention mechanism is replicated multiple times in a single layer, all with different weights (multi-head attention). This means that for a given model, there is $n_{layer} \times n_{head}$ attention results to consider, each attending to different parts of the input and enabling it to understand different linguistic features [38]. By averaging attention over a sequence of produced tokens, some heads can privy us to the importance of the sentences in the context, as shown in fig. 4.1. By learning custom tokens, one can also map multiple sub-answers (see appendix A), though those will be explored in future works.

LoRA. By only updating a small side matrix that can be applied back on the bigger initial matrix, Low Rank Adaptation (LoRA) [39] reduces the number of parameters that are trainable in a given model, thus reducing the training cost. We have used LoRA for all training procedures.

Quantization. Weights can be approximated by their quantized version, reducing their memory footprint through quantization. Via `bitsandbytes`[1], we used this technique in conjunction with LoRA (called QLoRA [40]) to fit our hardware constraints and improve reproducibility.

4.1.3 Method

Problem Statement. Given the triplet (Q, C, A), represented as sequences of tokens, we want to find the sequences (or rationales) $S \in C$ that best contribute to A. Meaning that, if we were to produce another answer A' from Q and S only, an ideal model M would find that $M(Q, C) = A \equiv A' = M(Q, S)$.

In the easiest case, A can be produced exactly using only a small subset of C, but in the general case, A might have to be inferred from the entire context. Nonetheless, we can assume that there exists a subset of C that is sufficient to produce A or that $A \equiv \{\}$.

Model. For the task outlined above, the output of the models can be formalized by $M(Q, C, A) = P$, where P is the predicted set of pertinent sentences (rationale) in the context and is represented as a set of tuples (I_{start}, I_{end}), respectively indexing the start and end of the sentences composing the predicted rationale. When models do not have access to A, the model becomes $M(Q, C) = A, P$. It is not always necessary to produce A, though our attention method finds it as a byproduct.

1 huggingface.co:bitsandbytes

Evaluation. For a single pair (P, S) of the predicted and reference rationale, we can define the Intersection over Union (IoU) score in eq. (4.1), with $|x|$ being the size of x. In this case, the size is computed by the sum of the lengths of each sub-element.

$$IoU(P,\ S) = \frac{|P \cap S|}{|P \cup S|} \tag{4.1}$$

4.1.4 Dataset

Definition. We utilize a CQA dataset because it inherently provides contextual information necessary for answering questions. By annotating the rationales required for answering each question, we have constructed a new dataset, referred to as the Rational Databricks Dolly (RDD) dataset.

Data source and filtering We specifically chose `databricks-dolly-15k` [41] as our base CQA dataset. We filtered the data points in the dataset by excluding those where the answer does not respond to the question using strictly the context. When little change was required to avoid discards (eg. deleting a sentence, adding a word,...), we applied it instead. This filtered dataset contains around 1.5K data-point.

Construction. From the filtered CQA dataset, each data-point has undergone human annotation to form our RDD dataset. The annotation process involves linking each (Q, A) pair in a data-point with the relevant sequences of tokens (the rationale), denoted as S, within the context C. To improve consistency in the annotation process, we have labeled entire sentences rather than segments of sentences. In cases where multiple questions existed within the same example, each sub-question has been labeled separately. This process can be described as in eq. (4.2). The tool we used to perform this task is `Doccano` [42].

$$\text{annotate}(Q, A, C) = [(Q_1, A_1, S_1), \ldots, (Q_N, A_N, S_N)] \tag{4.2}$$

The annotation process was done in this manner to potentially be used in an extension of the problem statement in which the model has to produce multiple sub-answers and explain the answer with multiple sub-rationales in context. For the rest of this work, we will consider that a data point is represented as a quadruplet (Q, A, C, S) where the input x is the triplet (Q, A, C) and the targeted output y is S.

4.2 Experiments

In this section, we will describe how the dataset has been used then define and explain the different methods tested for the RE task.

The dataset has been shuffled using the same random seed for all experiments to ensure consistency, and then divided into three sets: the training, validation and evaluation set. They represent respectively 80%, 10% and 10% of the original dataset. The training set will be used to train the methods, the validation set will be used to fine-tune the parameters, and finally the evaluation set will be used to compare their performances.

The methods are divided into three categories, all of which were able to run on our two Nvidia 2080ti.

The first category, referred to as the **Top-k methods**, consists of methods that retrieve the k most pertinent sentences according to a specific metric. The second category, referred to as the **threshold methods**, includes methods that retrieve all sentences above a certain threshold based on a given metric. The threshold is taken to be the best out of a sweep on the validation set. The last category consists of an **hybrid method** using both previous categories

4.2.1 Top-k methods

Top-k embedding similarity. The first method tested is an sentence embedding method based on LLMs pre-trained for Semantic Textual Similarity (STS). STS aims to determine the degree of similarity between two pieces of text. The hypothesis underlying this method is that the most pertinent sentences in the context will be those that are most similar to the answer. For this experiment setup, we utilized two pre-trained

LLMs: `Sentence-Bert`[43], one of the pioneering models for text similarity embedding based on LLM, and `SFR-Embedding-Mistral`[44], the current state-of-the-art (SOTA) of open-source model for STS tasks according to [45]. Cosine similarity is employed to compute the similarity between embedded sentences. Consequently, the model retrieves the k sentences with the highest similarity scores in the context, and can be formalized as in eq. (4.3)

$$P \equiv \textbf{Embedder}_{\text{Top-k}}(A,\, C) = \underset{s_j \text{ in } C}{\text{Top-k}}(\cos(\text{Embed}(A), \text{Embed}(s_j))) \tag{4.3}$$

Embed() is a function that takes a sentence as input and returns a vector $v \in \mathbb{R}^N$, with N being the size of the embedding (more details in appendix C).

Top-k N-Grams. The second method tested extends the TF-IDF N-Grams model [46] from a traditional text retrieval framework based on similarity scores to the RE task. This model involves embedding texts into vectors and computing the similarity between these embeddings using cosine similarity as for Top-k embedding. The hypothesis similarly is maintained. With this hypothesis, the RE task can be viewed as a text retrieval problem, where the answer A serves as the query and the sentences within the context C form the corpus of documents.

We also employ the classical IDF weighting method. The model retrieves the k sentences in the context with the highest similarity score. The model can be formalized as in eq. (4.4).

$$P \equiv \textbf{NG}_{\text{Top-k}}(A,\, C) = \underset{s \text{ in } C}{\text{Top-k}}(\cos(\text{TF-IDF}(A), \text{TF-IDF}(s_j))) \tag{4.4}$$

where s is a sentence of the context C. Top-k() returns the k largest element of a set. TF-IDF() is a function that takes a sentence as input and returns a vector $v \in \mathbb{R}^N$, with N being the size of the vocabulary. The TF-IDF() is fitted on the context C of the data-point, where each sentence is considered as a document (more details in appendix B).

4.2.2 Threshold methods

Instead of retrieving the k sentences with the highest similarity score in the context this model retrieves all the sentences with a similarity score higher than a certain threshold.

Threshold embedding similarity. This method can be formalized as in eq. (4.5).

$$P \equiv \textbf{Embedder}_{\text{Threshold}}(A,\, C) = \left\{ s \in C \;\middle|\; \cos\Big(\text{Embed}(A), \text{Embed}(s)\Big) > t \right\} \tag{4.5}$$

where t is the threshold.

Threshold N-Grams. This method can be formalized as in eq. (4.6).

$$P \equiv \textbf{NG}_{\text{Threshold}}(A,\, C) = \left\{ s \in C \;\middle|\; \cos\Big(\text{TF-IDF}(A), \text{TF-IDF}(s)\Big) > t \right\} \tag{4.6}$$

where t is the threshold.

LLM classifier. Inspired by the promising results of the fine-tuned LLM for text classification presented in [47] and [48], this experiment involves fine-tuning a LLM pre-trained for causal language modelling (cfr. section 4.1.2) to binary text classification. The objective of this fine-tuning is to determine whether a sentence in the context is part of the rationale or not. We have tested fine-tuning different pre-trained models with varying sizes of parameters to assess if more complex models can better handle this task. The different pre-trained LLMs tested are: `DistilBERT` [49], `RoBERTa-Base` [50], and `Gemma-2B` [51]. These methods can be formalized as in eq. (4.7).

$$P \equiv \mathbf{LLM}_{\text{Classifier}}(A,\, Q,\, C) = \left\{ s \in C \;\middle|\; l_1 > 0.5 \right\} \tag{4.7}$$

Where l_1 is the score of the positive label obtained by $classify(A,\, Q,\, s)$, representing the LLM classifier (more details in appendix D).

4.2.3 Hybrid method

This method combines both the threshold and the ranking. The hyper-parameter sweep was simply done with two dimensions.

LLM attention. By using the normalized vector of mean attention weights[2] for each generated token $Attn_{normalized}(j)$ as described in appendix A, we get the per sentence criterion of eq. (4.8).

$$\begin{aligned} P &\equiv LLM_{Attention}(Q, C) \\ &= \underset{s \text{ in } C}{\text{Top-K}} \left(\left\{ s \;\middle|\; \left(\frac{1}{|token(s)|} \sum_{j \in token(s)} tanh\left(Attn_{normalized}(j)\right) \right) > t \right\} \right) \end{aligned} \tag{4.8}$$

There are three variations. The first is the base model used directly as it was downloaded. The second is continually pre-trained [3] on the base CQA dataset by performing a standard causal language modelling training with Q, C and A always present concurrently. The third is a RL-tuned [4] version of the second, where the reward is α times the sum of the IoU score and the METEOR [52] metric. α has been set to 5 because it yielded better results.

Due to hardware constraints, we have limited the sizes of the examples to $|(Q, C)| < 2048$ and $|A| < 542$ for the total to respect $|(Q, C, A)| < 2600$. For the RL training, the values are 800, 64 and thus 864.

One distinguishing aspect of this method is that the model first generate the answer then attempt to find the rationale. It thus does not have access to the optimal answer and will likely perform worse in the cases where the quality of the answer lacks.

4.2.4 Results

As shown in fig. 4.2, the number of sentences in the context of each data point varies. We decided to split the data points into four categories based on the number of sentences in the context. This decision was made based on the hypothesis that the more sentences there are in the context, the more complex it is to identify the relevant sentence. Thus it is a valuable insight to provide.

The distribution is imbalanced, with far more examples having fewer sentences. Additionally, the three datasets exhibit quite similar distributions, which is not surprising given that the data points were randomly split in each dataset.

The results in table 4.1 summarize the performance of the different methods tested.

They indicate that classification methods outperform other approaches. The best classification method is the Gemma classifier, which shows a small improvement over the RoBERTa classifier. These findings suggest that larger models performs better at our small scale, or that models able to generate such answers are also more likely to find the correct rationales.

Additionally, the results demonstrate that the attention method performs better after training the LLM model on the training dataset despite not being trained for this metric in particular, highlighting a possible

2 Note that the attention is for a particular head of a particular layer in our case.
3 via Hugginface:Trainer
4 via HuggingFace:TRL:PPOTrainer

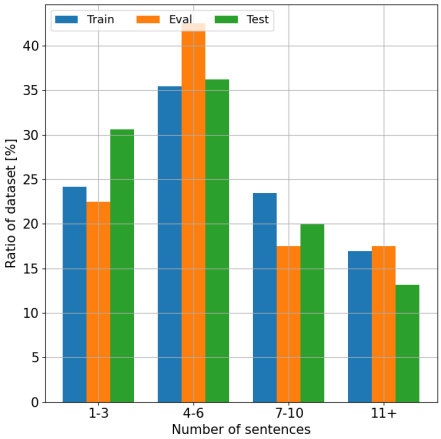


Figure 4.2: Data-point distribution by number of sentences in the context

correlation between rationale extraction and answer generation. Furthermore, RL training does continue improving upon the result, showing that we can target such a behavior during training.

The sweep of hyper-parameters has shown that Top-k methods perform best with small k values (i.e., 1 or 2). This is likely due to the skew of the dataset for smaller numbers of sentences (and rationales), as shown in fig. 4.2.

The results show also that very local model such as N-Grams performs quite well over the dataset contrasting more global model like Sentence-Bert-Large or SFR-Embedding-Mistral. These results show that the wording of the answer is similar to the relevant sentences within the context. This may be a particularity of this dataset.

As shown in fig. 4.3, as the number of sentences in the context increases, the performance of the models decrease. The figure also illustrates that most Top-k methods are capped around a 0.7 IoU score in the first group (1 à 3), likely because $k = 1$ restricts them from retrieving additional sentences needed for a higher score. In contrast, threshold methods do not have this limitation and can theoretically achieve an IoU score of 1 (i.e the maximum score). Hybrid methods are also theoretically capped but can allow a greater k and filter down with t.

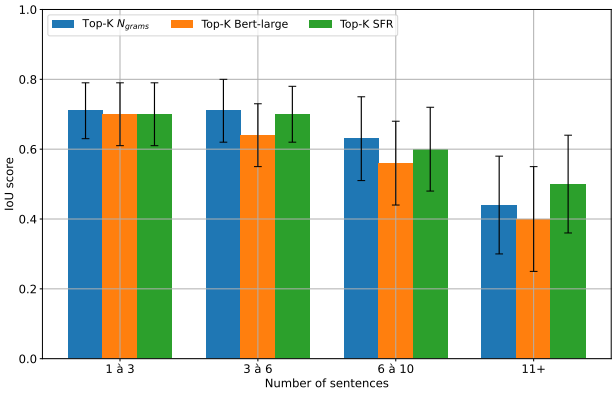
However, only the fine-tuned classifier outperforms the Top-k methods, this is likely because the number of sentences does not get big enough.

Lastly, the best method consistently perform better across all groups except for the third (6-10), for which the highest score is also achieved by a classifier.

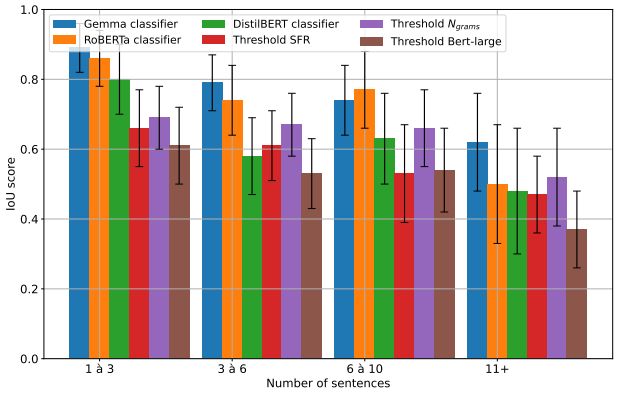
Model	Size	Average IoU
Top-K N_{grams} (k=1)	/	0.64 ± 0.05
Top-K Sentence-Bert-large (k=1)	109M	0.61 ± 0.05
Top-K SFR-Embedding-Mistral (k=1)	7.11B	0.65 ± 0.05
Gemma attention base(L=10, H=5, k=2, t=-0.005)	2.51B	0.53 ± 0.05
Gemma attention FT(L=10, H=7, k=2, t=0.015)	2.51B	0.60 ± 0.05
Gemma attention RL(L=10, H=7, k=2, t=0.015)	2.51B	0.62 ± 0.05
Threshold N_{grams} (t=0.25)	/	0.66 ± 0.05
Threshold Sentence-Bert-large (t=0.68)	109M	0.54 ± 0.06
Threshold SFR-Embedding-Mistral (t=0.72)	7.11B	0.59 ± 0.06
DistilBERT classifier	67M	0.64 ± 0.06
RoBERTa classifier	125M	0.75 ± 0.05
Gemma classifier	2.51B	$\mathbf{0.79 \pm 0.04}$

Table 4.1: Summary table of experiment results, including the number of parameters for each method (size) and the average IoU score obtained on the evaluation set, presented with a 95% confidence interval assuming a student-t distribution.

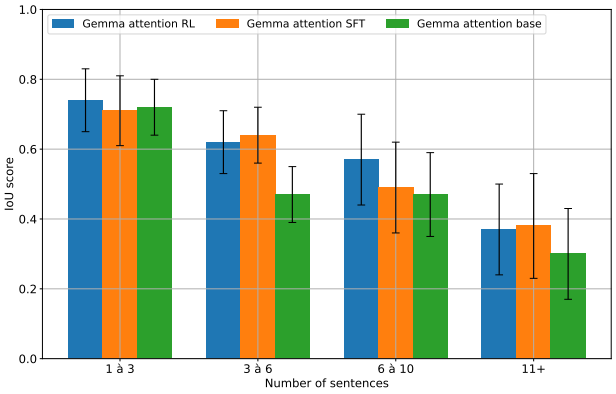
.



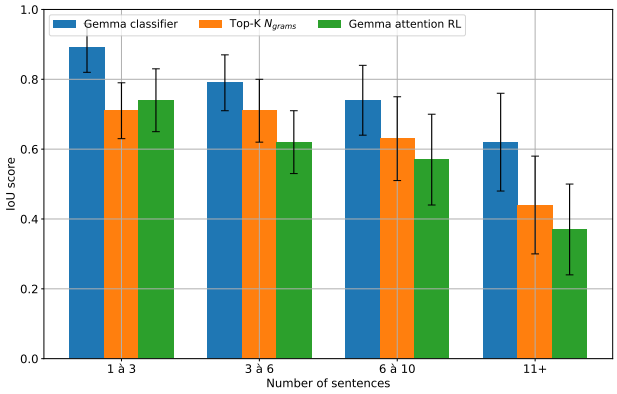
(a) Comparison of Top-K Methods



(b) Comparison of Threshold Methods



(c) Comparison of hybrid methods



(d) Comparison of the best methods across each categories

Figure 4.3: IoU scores of the different methods evaluated on the evaluation set. The bars represent the mean values, and the error bars indicate the 95% confidence interval for this mean scores assuming student-t distributions.

4.3 Limitations

The focus on attention methods rather than gradient based ones implies that this comparison cannot be extended easily to sub-quadratic [53, 54, 55, 56, 57], or attention-less [58, 59, 60, 61, 62] LLMs.

Our enhanced version of the dataset has only been annotated by us and thus may not have been reviewed impartially and/or in enough depth.

To quickly extend the dataset, we could have used existing datasets for rationale extraction in classification and procedurally generated appropriate outputs. However, this approach would decrease the variety of answers and the would be impact on performances has, to our knowledge, not yet been studied.

Despite our best attempt at exploring a diverse range of techniques and models, we are likely missing some that might perform better. Additionally, all the techniques and models we have used have been chosen to fit our hardware limitation, being two Nvidia 2080ti.

The tuning of hyper-parameters requiring a lot of compute, we have restricted the range of values to those we estimated would be most pertinent. The linear/grid search approach could also be revised to other optimization techniques.

We chose to explore attention-based methods over saliency and gradient-based methods to avoid back-propagation during generation (we might change course in future works) even if this choice is subject to debate [63].

4.4 Future works

In the produced dataset there have been numerous instances of questions that contained sub-questions. Following work will continue the approach while taking care of differentiating the sub-answers.

The answer quality has been overlooked and future work will try to improve it alongside our IoU score.

It may be interesting to see other approaches compete, like gradient-based methods, to see how the scores could be improved. Bigger models may also prove more accurate and the optimal trade-off could lead to more research.

4.5 Conclusion

In this chapter, we have constructed the Rationale Databricks Dolly dataset for the express purpose of improving rationale extraction in closed-domain question answering. We have studied a range of methods using the dataset to foster interest in the subject. These have been evaluated via our newly introduced IoU metric.

From our results, we have underlined the difficulty to reach satisfying results as the scale of the context grows. Nonetheless, we have found that classifier models could be used to find on average **79%** of the rationales on our dataset, paving the way to increasing fact-checking ability and remind the importance of explainability.

Such methods can be used down the line to improve generation. We have shown that it is possible to conjugate generation and rationale extraction, and that traditional training methods for answer quality did also affect positively the extraction ability. Finally, a direct targeting of both metrics is possible and also is beneficial.

Chapter 5

Conclusion

This work has explored the integration and utilization of large language models (LLMs) in industrial contexts, with a specific focus on Haulogy, a Belgian IT enterprise in the energy sector. By examining this practical example, we investigated the potential applications of LLMs in industrial settings, identified the challenges associated with their deployment, and proposed solutions to overcome these challenges.

Firstly, we formalized EnergyComm, a software solution developed by Haulogy, to better understand its functionality. Based on this formalization, we discussed potential improvements that LLMs could bring. The formalization highlighted that integrating AI and LLMs into EnergyComm would present challenges such as adapting to frequent market rule changes and ensuring the reliability of the solution. One promising approach is the retrieval-augmented generation (RAG) framework, which enables interaction with LLM chat models using a personalized knowledge dataset.

Secondly, we defined and tested the RAG framework. The proposed RAG prototype system showed promising results in its ability to answer specific questions about Haulogy. These capabilities are promising for speeding up the work of employees in industrial contexts. However, the system has weaknesses that could hinder its deployment. The main weaknesses are the lack of explainability of the answers and the lack of robustness. The design of the proposed prototype makes fact-checking the information used difficult due to the large volume of text. Another significant issue is the quality of the dataset, which is not always reliable and can lead to incorrect answers. Additionally, due to hardware constraints, the generator used in this prototype, Zephyr-7B, is small compared to the leading LLMs like GPT-4 or Gemini. This smaller size makes it less robust and less accurate than the larger models.

Finally, this work proposed an enhancement to closed-domain question answering (CQA). To achieve this, the Rationale Databricks Dolly dataset was constructed based on the databricks-dolly-15k dataset. A range of methods was tested on this dataset and evaluated using a newly introduced metric, the IoU score. The results highlighted the difficulty of achieving satisfactory outcomes as the scale of the context grows. Nonetheless, the classification method using the RoBERTa model was able to find 79% of the rationales on average in our dataset. These results are promising and could enhance fact-checking capabilities and improve explainability.

Moreover, such methods can also be used to improve text generation. This study demonstrated the possibility of combining generation and rationale extraction with attention-based methods. Indeed, traditional training procedures for answer quality also positively affect the extraction ability of these methods. Additionally, combining the IoU score with an NLP metric like METEOR as a reward for reinforcement learning training is feasible and beneficial, showing an improvement of 2% in the IoU score.

Bibliography

[1] OpenAI et al. *GPT-4 Technical Report*. 2024. arXiv: 2303.08774 [cs.CL].

[2] Gemini Team et al. *Gemini: A Family of Highly Capable Multimodal Models*. 2024. arXiv: 2312.11805 [cs.CL].

[3] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. arXiv: 2307.09288 [cs.CL].

[4] Tyna Eloundou et al. *GPTs are GPTs: An Early Look at the Labor Market Impact Potential of Large Language Models*. 2023. arXiv: 2303.10130 [econ.GN].

[5] Patrick Lewis et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. 2021. arXiv: 2005.11401 [cs.CL].

[6] Wikipedia. *Business rules engine*. URL: https://en.wikipedia.org/wiki/Business_rules_engine#:~:text=Rules%20engines%20or%20inference%20engines,the%20need%20for%20IT%20intervention (visited on 08/23/2023).

[7] F.Marchioni. *What is a rule engine ?* URL: https://www.masterthebоss.com/bpm/drools/what-is-a-rule-engine/#:~:text=A%20rule%20engine%20combines%20a,which%20we%20assume%20rarely%20changes (visited on 08/23/2023).

[8] Team EnergyComm. *EnergyComm FR Kernel Rules - V8.0*. Haulogy. Braine-le-Comte, Belguim, 2023.

[9] Neel Kant et al. *Practical Text Classification With Large Pre-Trained Language Models*. 2018. arXiv: 1812.01207 [cs.CL].

[10] Yang Bai and Daisy Zhe Wang. *More Than Reading Comprehension: A Survey on Datasets and Metrics of Textual Question Answering*. 2022. arXiv: 2109.12264 [cs.CL].

[11] Philip Feldman, James R. Foulds, and Shimei Pan. *Trapping LLM Hallucinations Using Tagged Context Prompts*. 2023. arXiv: 2306.06085 [cs.CL].

[12] Yiheng Liu et al. "Summary of ChatGPT-Related research and perspective towards the future of large language models". In: *Meta-Radiology* 1.2 (2023), p. 100017. ISSN: 2950-1628. DOI: https://doi.org/10.1016/j.metrad.2023.100017. URL: https://www.sciencedirect.com/science/article/pii/S2950162823000176.

[13] Guilherme Penedo et al. *The RefinedWeb Dataset for Falcon LLM: Outperforming Curated Corpora with Web Data, and Web Data Only*. 2023. arXiv: 2306.0111 [cs.CL].

[14] Rico Sennrich, Barry Haddow, and Alexandra Birch. *Neural Machine Translation of Rare Words with Subword Units*. 2016. arXiv: 1508.07909 [cs.CL].

[15] Xinying Song et al. *Fast WordPiece Tokenization*. 2021. arXiv: 2012.15524 [cs.CL].

[16] Taku Kudo and John Richardson. *SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing*. 2018. arXiv: 1808.06226 [cs.CL].

[17] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[18] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. arXiv: 1907.11692 [cs.CL].

[19] Arvind Neelakantan et al. *Text and Code Embeddings by Contrastive Pre-Training*. 2022. arXiv: 2201.10005 [cs.CL].

[20] Shitao Xiao et al. *C-Pack: Packaged Resources To Advance General Chinese Embedding*. 2024. arXiv: 2309.07597 [cs.CL].

[21] IBM. *What is a vector database?* URL: https://www.ibm.com/topics/vector-database#:~:text=Vector%20databases%20are%20rapidly%20growing,models%20with%20relevant%20business%20data. (visited on 05/26/2024).

[22] Lewis Tunstall et al. *Zephyr: Direct Distillation of LM Alignment*. 2023. arXiv: 2310.16944 [cs.LG].

[23] Ming Li et al. *Superfiltering: Weak-to-Strong Data Filtering for Fast Instruction-Tuning*. 2024. arXiv: 2402.00530 [cs.CL].

[24] Haiyan Zhao et al. "Explainability for Large Language Models: A Survey". In: *ACM Transactions on Intelligent Systems and Technology* 15.2 (2024), pp. 1–38.

[25] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. ""Why Should I Trust You?": Explaining the Predictions of Any Classifier". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*. Ed. by Balaji Krishnapuram et al. ACM, 2016, pp. 1135–1144. DOI: 10.1145/2939672.2939778. URL: https://doi.org/10.1145/2939672.2939778.

[26] Ilias Chalkidis et al. "Paragraph-level Rationale Extraction through Regularization: A case study on European Court of Human Rights Cases". In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Ed. by Kristina Toutanova et al. Online: Association for Computational Linguistics, June 2021, pp. 226–241. DOI: 10.18653/v1/2021.naacl-main.22. URL: https://aclanthology.org/2021.naacl-main.22.

[27] Satyapriya Krishna et al. "Post Hoc Explanations of Language Models Can Improve Language Models". In: *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*. Ed. by Alice Oh et al. 2023. URL: http://papers.nips.cc/paper%5C_files/paper/2023/hash/ce65173b994cf7c925c71b482ee14a8d-Abstract-Conference.html.

[28] Machel Reid et al. *Gemini 1.5: Unlocking Multimodal Understanding across Millions of Tokens of Context*. Mar. 2024. arXiv: 2403.05530 [cs]. (Visited on 03/13/2024).

[29] Hao Liu, Matei Zaharia, and Pieter Abbeel. "Ring Attention with Blockwise Transformers for Near-Infinite Context". In: *CoRR* abs/2310.01889 (2023). DOI: 10.48550/ARXIV.2310.01889. arXiv: 2310.01889. URL: https://doi.org/10.48550/arXiv.2310.01889.

[30] Freda Shi et al. "Large Language Models Can Be Easily Distracted by Irrelevant Context". In: *Proceedings of the 40th International Conference on Machine Learning*. PMLR, July 2023, pp. 31210–31227. (Visited on 05/10/2024).

[31] Ines Arous et al. "MARTA: Leveraging Human Rationales for Explainable Text Classification". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.7 (May 2021), pp. 5868–5876. DOI: 10.1609/aaai.v35i7.16734. URL: https://ojs.aaai.org/index.php/AAAI/article/view/16734.

[32] Andrew Slavin Ross, Michael C. Hughes, and Finale Doshi-Velez. "Right for the Right Reasons: Training Differentiable Models by Constraining their Explanations". In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*. Ed. by Carles Sierra. ijcai.org, 2017, pp. 2662–2670. DOI: 10.24963/IJCAI.2017/371. URL: https://doi.org/10.24963/ijcai.2017/371.

[33] Aaron Chan et al. "UNIREX: A Unified Learning Framework for Language Model Rationale Extraction". In: *Proceedings of the 39th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri et al. Vol. 162. Proceedings of Machine Learning Research. PMLR, July 2022, pp. 2867–2889.

[34] Pooya Moradi, Nishant Kambhatla, and Anoop Sarkar. "Measuring and Improving Faithfulness of Attention in Neural Machine Translation". In: *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. Ed. by Paola Merlo, Jorg Tiedemann, and Reut Tsarfaty. Online: Association for Computational Linguistics, Apr. 2021, pp. 2791–2802. DOI: 10.18653/v1/2021.eacl-main.243.

[35] Max Glockner, Ivan Habernal, and Iryna Gurevych. "Why do you think that? Exploring Faithful Sentence-Level Rationales Without Supervision". In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Ed. by Trevor Cohn, Yulan He, and Yang Liu. Online: Association for Computational Linguistics, Nov. 2020, pp. 1080–1095. DOI: 10.18653/v1/2020.findings-emnlp.97. URL: https://aclanthology.org/2020.findings-emnlp.97.

[36] Brihi Joshi et al. "ER-test: Evaluating Explanation Regularization Methods for Language Models". In: *Findings of the Association for Computational Linguistics: EMNLP 2022, Abu Dhabi, United Arab*

Emirates, December 7-11, 2022. Ed. by Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang. Association for Computational Linguistics, 2022, pp. 3315–3336. DOI: 10.18653/V1/2022.FINDINGS-EMNLP.242.

[37] Jorge Ramírez et al. "Understanding the Impact of Text Highlighting in Crowdsourcing Tasks". In: *Proceedings of the Seventh AAAI Conference on Human Computation and Crowdsourcing, HCOMP 2019, Stevenson, WA, USA, October 28-30, 2019*. Ed. by Edith Law and Jennifer Wortman Vaughan. AAAI Press, 2019, pp. 144–152. DOI: 10.1609/HCOMP.V7I1.5268. URL: https://doi.org/10.1609/hcomp.v7i1.5268.

[38] Kevin Clark et al. *What Does BERT Look At? An Analysis of BERT's Attention*. June 2019. arXiv: 1906.04341 [cs]. (Visited on 04/09/2024).

[39] Edward J. Hu et al. "LoRA: Low-Rank Adaptation of Large Language Models". In: *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[40] Tim Dettmers et al. *QLoRA: Efficient Finetuning of Quantized LLMs*. 2023. arXiv: 2305.14314 [cs.LG].

[41] Mike Conover et al. *Free Dolly: Introducing the World's First Truly Open Instruction-Tuned LLM*. 2023. URL: https://www.databricks.com/blog/2023/04/12/dolly-first-open-commercially-viable-instruction-tuned-llm (visited on 02/05/2024).

[42] Hiroki Nakayama et al. *doccano: Text Annotation Tool for Human*. Software available from https://github.com/doccano/do 2018. URL: https://github.com/doccano/doccano.

[43] Nils Reimers and Iryna Gurevych. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. 2019. arXiv: 1908.10084 [cs.CL].

[44] Salesforce AI Research. *SFR-Embedding-Mistral: Enhance Text Retrieval with Transfer Learning*. URL: https://blog.salesforceairesearch.com/sfr-embedded-mistral/ (visited on 05/28/2024).

[45] Huggingface. *mteb-leaderboard*. URL: https://huggingface.co/spaces/mteb/leaderboard (visited on 05/28/2024).

[46] Gerard Salton and Christopher Buckley. "Term-weighting approaches in automatic text retrieval". In: *Information Processing & Management* 24.5 (1988), pp. 513–523. ISSN: 0306-4573. DOI: https://doi.org/10.1016/0306-4573(88)90021-0. URL: https://www.sciencedirect.com/science/article/pii/0306457388900210.

[47] Xiaofei Sun et al. *Text Classification via Large Language Models*. 2023. arXiv: 2305.08377 [cs.CL].

[48] Youngjin Chae and Thomas Davidson. *Large Language Models for Text Classification: From Zero-Shot Learning to Fine-Tuning*. 2023. DOI: 10.31235/osf.io/sthwk.

[49] Victor Sanh et al. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. 2020. arXiv: 1910.01108 [cs.CL].

[50] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. arXiv: 1907.11692 [cs.CL].

[51] Gemma Team et al. *Gemma: Open Models Based on Gemini Research and Technology*. 2024. arXiv: 2403.08295 [cs.CL].

[52] Satanjeev Banerjee and Alon Lavie. "METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments". In: *Proceedings of the Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization@ACL 2005, Ann Arbor, Michigan, USA, June 29, 2005*. Ed. by Jade Goldstein et al. Association for Computational Linguistics, 2005, pp. 65–72.

[53] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. "Reformer: The Efficient Transformer". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[54] Jiayu Ding et al. "LongNet: Scaling Transformers to 1, 000, 000, 000 Tokens". In: *CoRR* abs/2307.02486 (2023). DOI: 10.48550/arXiv.2307.02486. arXiv: 2307.02486.

[55] Xuezhe Ma et al. "Mega: Moving Average Equipped Gated Attention". In: *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[56] Krzysztof Marcin Choromanski et al. "Rethinking Attention with Performers". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[57] Kaiqiang Song et al. *Zebra: Extending Context Window with Layerwise Grouped Local-Global Attention*. Dec. 2023. DOI: 10.48550/arXiv.2312.08618. arXiv: 2312.08618 [cs]. (Visited on 12/15/2023).

[58] Albert Gu and Tri Dao. *Mamba: Linear-Time Sequence Modeling with Selective State Spaces*. Dec. 2023. DOI: 10.48550/arXiv.2312.00752. arXiv: 2312.00752 [cs]. (Visited on 12/12/2023).

[59] Shuangfei Zhai et al. "An Attention Free Transformer". In: *CoRR* abs/2105.14103 (2021). arXiv: 2105.14103.

[60] Soham De et al. *Griffin: Mixing Gated Linear Recurrences with Local Attention for Efficient Language Models*. Feb. 2024. DOI: 10.48550/arXiv.2402.19427. arXiv: 2402.19427 [cs]. (Visited on 03/13/2024).

[61] Bo Peng et al. "RWKV: Reinventing RNNs for the Transformer Era". In: *Conference on Empirical Methods in Natural Language Processing* (2023). DOI: 10.48550/arXiv.2305.13048.

[62] Maximilian Beck et al. "xLSTM: Extended Long Short-Term Memory". In: *arXiv preprint arXiv: 2405.04517* (2024).

[63] Jasmijn Bastings and Katja Filippova. "The Elephant in the Interpretability Room: Why Use Attention as Explanation When We Have Saliency Methods?" In: *BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP* (2020). DOI: 10.18653/V1/2020.BLACKBOXNLP-1.14.

Appendix A

Attention representation

Single-rationale. The displayed colors of fig. 4.1 are simply the projection of the obtained values onto RGB where all colors are of equal values.

In particular, we obtain the normalized values per token $Attn_{normalized}(j)$ by eq. (A.1), then eq. (A.2) with $\epsilon = 1e - 8$ added for stability.

$$Attn(j) = \begin{cases} \frac{1}{|token(A)|} \sum_{i \in token(A)} attn(i,j) & j \in (token(C) \backslash token(A)) \\ 0 & otherwise \end{cases} \tag{A.1}$$

$$Attn_{normalized}(j) = \begin{cases} \frac{Attn(j) - mean(Attn)}{std(Attn) + \epsilon} & j \in (token(C) \backslash token(A)) \\ 0 & otherwise \end{cases} \tag{A.2}$$

Multi-rationale. By fine-tuning a model on our RDD dataset, we can provide markers for rationales relating to sub-questions, thereby disambiguating the tokens selected to do the averaging and allow specific references to be made. Applying the same method as for single-rationale but varying channels depending on the rationale index, we can have the representation on fig. A.1.

Figure A.1: Segmented average attention weights over a generation by google/gemma-2b, colorized (red and yellow for first and second citation respectively).

Appendix B

TF-IDF Technique and N-Grams

In this section, we delve into the TF-IDF technique and its relationship with N-Grams. TF-IDF encapsulates the significance of terms within documents concerning a corpus, thereby facilitating tasks such as information retrieval and text mining. In our context, terms can be individual words (uni-grams) or combinations of neighbors words (N-Grams), with documents represented as sentences of the context or a answer text.

TF-IDF operates by transforming documents into numerical vectors, where each dimension corresponds to a term in the vocabulary. The value of each dimension represents the TF-IDF score of the corresponding term in the document. This score, derived from the Term Frequency (TF) and Inverse Document Frequency (IDF) values, highlights terms that are prevalent within a document but rare across the corpus, thereby underlining their significance within that document.

The TF component quantifies the frequency of a term within a document by dividing the number of occurrences of a term by the total number of terms in the document:

$$\mathrm{TF}(t_i, D) = \frac{f_t}{T} \tag{B.1}$$

Where T is the total number of terms in the documents D and f_t the frequency of the term i (t_i) in the document T.

Conversely, the IDF component measures the significance of a term across the entire corpus. It is calculated using the following formula:

$$\mathrm{IDF}(t_i) = \log\left(\frac{N}{n}\right) + 1 \tag{B.2}$$

Where, N represents the total number of documents, and n represents the number of documents containing the term t_i. Additionally, the addition of "1" to the IDF formula ensures that terms with zero IDF, i.e., terms present in all documents within a training set, are not entirely disregarded.

Finaly the TF-IDF score of a term in the document D is calculated as follows:

$$\text{TF-IDF}_{\text{score}}(t_i, D) = \mathrm{TF}(t_i, D) \times \mathrm{IDF}(t_i) \tag{B.3}$$

The construction of the TF-IDF vector of a document based on an reference corpus is described by the following pseudo-code:

Algorithm 1 TF-IDF Fit and Transform

Inputs:
C : The corpus composed of n documents D_i
Q : The query document

Outputs:
E_Q : The embedding vector of the query document
EM : The embedding matrix of the corpus composed of n horizontal embedding vectors

```
# Create a set to store the vocabulary of the corpus (i.e all the unique terms in the corpus)
V = ( )

# Construct the vocabulary of the corpus by iterate over each document of the corpus
for D in C do
    # split () splits the document D into terms. e.g [t_1, ..., t_n] = split(D)
    T = split(D)
    V = V ⋃ T
end for

# Compute the term frequency matrix of the corpus
# First create a matrices of size V × C fill with zeros
EM = zeros (length (V), length (C))
for i, D in enumerate (C) do
    Word_list = split (D)
    for j, word in enumerate(V) do
        # count (a, b) is a function that count the occurrence of a in list b
        EM[i, j] = count (word, Word_list)/length (Word_list)
    end for
end for

# Construct the IDF vector over the corpus
# First create a vector of size C fill with zeros
IDF = zeros (length(C))
for i in range (length (V)) do
    # count ("0 >", M[i, :]) is a function that counts
    # the number of non-zero elements in the i^th column of M.
    IDF[ i ] = log (count ("0 >", M[i, :]) / length (C)) + 1
end for

for i, _ in enumerate (C) do
    for j, _ in enumerate (V) do
        EM[i, j] = EM[i, j] * IDF[ i ]
    end for
end for

# Transform the Query document
E_Q = zeros(length(V), 1)
Query_words = split(Q)
for i, word in enumerate(V) do
    # count (a, b) is a function that count the occurrence of a in list b
    E_Q[i] = count (word, Query_words)/length (Query_words)
    E_Q[i] = E_Q[i] * IDF[ i ]
end for
```

Appendix C

Embedding based on LLM

In this section, we delve into the embedding method based on pre-trained LLMs. Certain LLMs have been specifically trained for a task called **semantic textual similarity (STS)**. This task involves determining the degree of similarity between two texts. Sentence similarity models convert input texts into vectors of fixed size (embedding) that capture semantic information and calculate how close (or similar) they are to each other. This task finds particular utility in information retrieval and clustering/grouping.

To generate the embedding of a sentence (or text) using this LLM, the most commonly employed approach is to either average the final hidden vector (before the classification layer of a classical causal Transformer LLM) of each token in the sentence (or text), or simply pool the final hidden vector of the special first token of every input (the [CLS] token). For our experiment, we have chosen to use the final hidden vector of the [CLS] token.

The pre-trained LLMs that we have selected are **SBERT** and **SFR-Embedding-Mistral**. The embedding base on LLM can be formalized by the following algorithm:

Algorithm 2 Embedded a document using pre-trained LLM for sentence similarity

Inputs:
D : the textual document
LLM: a pre-trained LLM model
$Tokenizer$: a Tokenizer compatible with the LLM

Outputs:
E : The embedding of the document

$tokens \leftarrow Tokenizer.tokenize(D)$
Pool the last hidden vector of the LLM for the CLS token
$last\_layer \leftarrow LLM(tokens)$
$E \leftarrow last\_layer$.CLS

Appendix D

LLM classifier

With the rise of LLMs and the transformer architecture for text generation, the utilization of transformers for text classification has been extensively explored in many papers. The architecture of the classifier used for our task has been inspired by previous works such as [47] and [48]. In this section, we will detail the architecture and the pipeline used to fine-tune the pre-trained LLMs for our experiment.

The input to the classifier consists of a concatenation of a context window around the sentence to be classified (w), the answer text (A), and the question text (Q). The context window includes the sentence to be classified, as well as the preceding and following sentences, if they exist. An example of the formatted input is presented in fig. D.2.

D.1 Architecture

Firstly the architecture of our LLM classifier is the same for the pre-trained model and was generated with the help of the class `AutoModelForSequenceClassification` from the library `Transformers`. As shown on fig. D.1, our classifier is simply formed by the pre-trained LLM from which we poll the last hidden vector and use this vector as input of a linear layer with a output size two (the number of label). Moreover, the tokenizer use in each classifier will be the standard tokenizer associated to pre-trained LLM.

D.2 Training data

To train the model, we utilized the training set. It's important to note that this model classifies sentences individually within the context. Thus, one data point from the training set may correspond to multiple sentences to classify, resulting in multiple instances of classification. Therefore, the original RDD training set with an original size of N has been extended to size M, where $M = \sum_i^N |C_i|$, and $|C_i|$ represents the number of sentences in the context of data point i. This dataset extension process is formalized by algorithm 3.

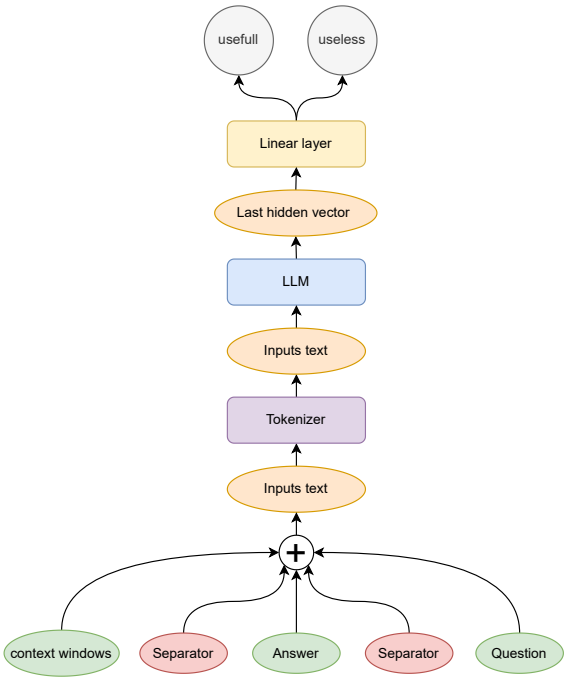


Figure D.1: Block diagram of the architecture of the LLM classifier

sentence to classify:
This small mansion has medieval origns and is surrounded by a large landscaped park.<!>The present building was constructed in 1634 by Evan Edwards, a member of a well established Flintshire family which traced its descent from the Welsh king Hywel Dda.<!> He most likely incorporated an older medieval house into the north wing of the current building.
Answer:
Rhual was constructed in 1634 by Evan Edwards
Question:
When was Rhual constructed?

(a) Positive example

Context windows:
Rhual is a Grade I listed building in Flintshire. <!>This small mansion has medieval origns and is surrounded by a large landscaped park.<!> The present building was constructed in 1634 by Evan Edwards, a member of a well established Flintshire family which traced its descent from the Welsh king Hywel Dda.
Answer:
Rhual was constructed in 1634 by Evan Edwards
Question:
When was Rhual constructed?

(b) Negative example

Figure D.2: Examples of the formatted input feed to the LLM classifier. Notice that sentence to classify is highlighted in orange within the context windows.

Algorithm 3 Extension of the RDD set to classification set

Inputs:
RDD : The original context and citation set

Outputs:
CS : Classification set

$CTS \leftarrow [\,]$
for x in range(len(RDD)) **do**
    $C \leftarrow x.context$
    $Q \leftarrow x.question$
    $A \leftarrow x.answer$
    $CI \leftarrow x.citations$
    $S \leftarrow splits(C)$
    for i in range(len(S)) **do**
        if $i == 0$ **then**
            $w \leftarrow S[\,i\,] + S[\,i+1\,]$
        else if $i ==$ len(S) **then**
            $w \leftarrow S[\,i-1\,] + S[\,i\,]$
        else
            $w \leftarrow S[\,i-1\,] + S[\,i\,] + S[\,i+1\,]$
        end if
        $text \leftarrow w + A + Q$
        $y \leftarrow 0$
        if s in CI **then**
            $y \leftarrow 1$
        end if
        $CTS.append(\{text,\ y\})$
    end for
end for

D.3 Fine-tuning

The fine-tuning pipeline is consistent across all pre-trained LLMs tested. The LLM weights and the new linear layer are fine-tuned over the training set, while the validation set is used to assess the generalization of the LLM and observe the impact of hyperparameter tuning. Moreover, the pre-trained LLMs have been fine-tuned using QLoRa [40], the state-of-the-art in Parameter-Efficient Fine-Tuning (PEFT) techniques, which enables training of selected models on a single Nvidia 2080Ti GPU.
The fine-tuning process has been done with the libraries `transformers` and `peft` from HuggingFace.

During training, the choice of the loss function is the cross-entropy loss, and evaluation and validation are based on accuracy. This selection is based on the fact that the model classifies individual sentences within the context, rendering the Intersection over Union (IoU) metric meaningless in this context. However, during the evaluation phase, the IoU metric will be used, similar to other models. This is because during evaluation, the model will be assessed based on its ability to correctly select sentences within the context of a data point.

All models are fine-tuned over 20 epochs, and the saved checkpoint is the one at which the model achieved the best validation accuracy throughout the training process. This approach is taken because the training process can be unstable and the big models are prone to overfitting.

Hyperparameter tuning
Depending on the size of the pre-trained LLM, the hyperparameters have been fine-tuned to varying degrees due to the longer training time required for larger models. Hyperparameter selection is primarily based on empirical results, with a focus on the following order: learning rate, LoRa rank, alpha LoRa, and LoRa

dropout. The final parameter used for all fine-tuned models are the following: Learning rate=5e-5, LoRa rank=4, alpha LoRa=4, LoRa dropout=0.1,