UNIVERSITÁ DEGLI STUDI DI MILANO-BICOCCA

Department of Computer Science, Systems and Communication

Master's Degree Program in Data Science

ON LLMs FOR SYNTHETIC STRUCTURED DATA GENERATION

Thesis Supervisor: Mirko Cesarini

Master's Thesis of:

Alessio Giannelli

ID Number 846586

Academic Year 2022/2023

Abstract

This paper, encapsulates the journey I have been on over the past few months toward exploring large language models, in which I have had the opportunity to experiment with processes of data acquisition, data extraction, data generation, and prompt engineering.

In the first project, on which the title of my thesis is focused, I analyzed the ability of LLMs to replicate (synthetize) structured data, with the goal of generating data quality rule configurations. The data processed were in Json format, and thanks to prompt engineering and langchain extensions, I was able to integrate the first version of a chatbot within the data quality tool ensuring the privacy of our customers data.

In the second project, which is described more briefly, I explored the ability of llms in extracting information from complex documents. The data, from data lakes, were preprocessed through OCR and segmentation techniques. After that with the help of LLMs and prompt engineering, a process of information extraction from the documents was implemented.

The models I intentionally chose to engage belong to the most revolutionary companies in the genAI landscape: GPT3.5 - turbo by OpenAI and Calude V2.1 and Claude Instant by Anthropic. In both projects, results were achieved that highlight the performance of LLMs and their revolutionary nature; however, the thesis aims not only at highlighting the strengths of these models, but also at raising awareness of these new technologies.

Indice

1 Introduction **1**

1.1 Context 1

1.2 Goals 1

1.3 Contribution 2

1.4 Thesis Structure 3

2 State of the Art **4**

2.1 Data 4

2.1.1 Existing paradigms for processing structured data 5

2.1.2 Existing paradigms for processing unstructured data 5

2.1.3 CAP Theorem 6

2.2 History of NLP 7

2.2.1 Turing Test 9

2.2.2 First Chatbot 9

2.2.3 From POS to Machine Learning 10

2.2.4 Deep Learning steps in 11

2.2.5 Nowdays 11

2.3 NLP Techniques for Synthetic Data 13

2.3.1 Artificial Neural Networks 13

2.3.2 Architecture and Components 14

2.3.3 RNN and LSTM 15

2.3.4 Word Embeddings 17

2.3.5 GAN 18

2.3.6 Data Masking 19

2.3.7 Trasformers 19

2.4 Cosine Similarity 22

2.4.1 Definition of Cosine Similarity 22

2.4.2 Applicability in LLM Output Validation . . . 22
2.5 Large Language Models . . . 23
2.5.1 Fine-tuning . . . 24
2.5.2 Fine-tuning Techniques . . . 25
2.5.3 GPT - OpenAI . . . 26
2.5.4 Claude V2 - Anthropic . . . 27
2.6 Technologies, Ecosystems and Programming Language . . . 28
2.6.1 Prompt Engineering . . . 28
2.6.2 Python . . . 30
2.6.3 Amazon SageMaker . . . 31
2.6.4 Amazon Bedrock . . . 31
2.6.5 Langchain . . . 32

3 Implementation **35**
3.1 Alertable . . . 36
3.1.1 AlerTable Architecture . . . 36
3.2 AlerTable Components . . . 37
3.3 Datasource Import . . . 38
3.4 Data Quality Rule Configuration . . . 39
3.4.1 Configurazione Check . . . 39
3.4.2 Configurazione Alert . . . 43
3.5 Data Preparation and Storage . . . 45
3.5.1 Dataset . . . 45
3.5.2 Datasource Import - Json . . . 46
3.5.3 Check - Json . . . 47
3.5.4 Alert - Json . . . 47
3.6 Models . . . 48
3.7 Kor - Extract Structured Data From Text . . . 49
3.8 Input - Prompt . . . 53
3.9 Output . . . 53

4 Evaluation **55**

4.1 Synthetic Data Validation 55

4.2 Performance Comparison 59

5 Parallel Project **62**

5.1 Description 62

5.2 Preprocessing 63

5.2.1 OCR 64

5.2.2 Data Cleaning and Segmentation 64

5.3 Prompt Engeenering and Extraction 65

5.4 Post-Processing 66

5.5 Evaluation 66

6 Conclusion and Future Developments **68**

6.1 Conclusion 68

6.2 Flowchart 72

6.3 Future Development 73

Appendix A: Python Libraries **73**

Appendix B: Check Synthetic Generation **75**

Appendix C: Alert Synthetic Generation **81**

Bibliography **85**

List of Figures

2.1 CAP Theorem 7
2.2 Rosetta Stone 8
2.3 Eliza Chatbot 10
2.4 Chatbot Timeline 12
2.5 1 Million user 12
2.6 Biological neuron vs Artificial neural network 14
2.7 Architecture of an Artificial Neural Network with a single hidden layer 15
2.8 Recurrent Neural Network Architecture 16
2.9 Recurrent Neural Network e Forward Neural Network Architectures 17
2.10 Word Embeddings Rapresentation 18
2.11 Trasformer - Model Architecture 20
2.12 LLMs Timeline 24
2.13 LLMs Fine-tuning Process 25
2.14 Prompt Operation 28
2.15 Langchain Application Lifecycle 34

3.1 AlertTable Architecture 37
3.2 List of datasource 38
3.3 Import Function 38
3.4 Check Configuration 40
3.5 Alert Configuration 44
3.6 Datasource Import Json Structure 47
3.7 Check Json Structure 47
3.8 Alert Json Structure 48
3.9 GPT3.5 - turbo, model parameters 48
3.10 Claude V2.1, model parameters 48
3.11 Check Prompt - Kor Schema part1 50

3.12 Check Prompt - Kor Schema part2 50
3.13 Check Prompt - Kor Schema examples 51
3.14 Alert Prompt - Kor Schema part1 52
3.15 Alert Prompt - Kor Schema examples 52
3.16 Input Model - Check Example 53
3.17 Input Model - Alert Example 53
3.18 Output Model - Check Example 54
3.19 Output Model - Alert Example 54

4.1 Performance Comparison - Check 60
4.2 Performance Comparison - Alert 61

5.1 Page Example 63

6.1 Performance Comparison - Data Quality Rule: Check Component 69
6.2 Performance Comparison - Data Quality Rule: Alert Component 69
6.3 Goals Achieved - Flowchart 72

1. Introduction

1.1 Context

I would like to extend my sincere appreciation for the invaluable support and contributions provided by the company that has accompanied me throughout this academic journey: Reply S.p.A., the holding company, and particularly Data Reply, where I had the privilege of completing an internship and conducting the majority of the work detailed in this thesis.

Reply is a multinational consulting firm specializing in various branches of Information Technology (IT). With a workforce of over 14,000 employees, Reply operates through a network structure comprising more than 150 specialized companies across different sectors. Founded in Turin on July 2, 1996, by Mario Rizzante, an entrepreneur who transitioned from the FIAT group, Reply has evolved into an empire spanning 38 countries, listed on the stock exchange, with a market capitalization exceeding 1 billion euros. Data Reply, a factory of the Reply group, offers a comprehensive range of advanced analytics and AI-powered data services. Serving diverse industries such as finance, insurance, transportation, healthcare, and manufacturing, Data Reply excels in designing and implementing data and analytics platforms, as well as executing artificial intelligence and machine learning projects. Leveraging predictive analytics techniques, Data Reply empowers businesses to make informed decisions [47].

As an integral part of the Reply Group, Data Reply benefits from synergies and expertise across the entire group, enabling the delivery of innovative solutions tailored to clients' needs.

1.2 Goals

In recent years, interest in NLP (acronym for Natural Language Processing) has grown exponentially, with big tech, investing large amounts of money. In particular, a lot of attention arose on models called LLM (Large Language Model), and specifically, ChatGPT created by OpenAI. It became clear in the first few months of 2023 how these models might affect the everyday lives of ordinary users. However, the primary goal I'd like to achieve is understanding how they can be

applied for commercial purposes. I focused on automating tedious and repetitive processes for customers, saving them both time and money. It's also crucial to determine which models can be used on which platforms to ensure data protection, as many exciting models are being released, but some of them raise some privacy concerns since the data ingested by them might be stored and used for further training the models. Some customers of course wouldn't accept that. The thesis began with a review of the state of the art, outlining the tools I employed and the reasons for their selections. This work is centered on testing the data quality of differet. The goal of the thesis focuses on how LLMs can perform quality testing on customer data, used natural language, ensuring privacy and avoiding the use of complex platforms.

Specifically, the ultimate goal of the project aims at the integration of an LLM within the AlerTable tool, which will be described in the following chapters. The pre-posed goal is to replicate a JSON structure that identifies the configuration of a data quality rule using LLMs. The data quality rule consists mainly of two components Check and Alert (they will be described later in detail). The Model starting from a natural language input and will have to cope with the following challenges: generate the output ensuring that the data inside the datasets are not disclosed; must be able to test multiple datasources; configure multiple checks that may involve multiple datasources; Launch multiple alerts involving multiple checks.

1.3 Contribution

The contribution of my thesis is to demonstrate the power of Large Language Models in creating structured and semistructured output data from natural language input. All while ensuring the privacy of our customers data. The selected algorithms that will help achieve the above goals are two: GPT3.5 - turbo, Claude V2.1 and Claude Instant. The first is an advanced version of the famous ChatGPT, a product pioneered by the OpenAI house and known to anyone since 2021 for paving the way for Generative Artificial Intelligence. The second one, on the other hand, was launched by the company Anthropic, created not surprisingly by former OpenAI employees. The techniques that I will approach alongside these two formidable genAI models, to get to replicate(synthesize) the json structures that define the configuration of data quality rules, will be many. Among these it

is necessary to highlight prompt engineering, which is essential to make sure that model responses are consistent and contextualized, and secondly, it is necessary to mention the adoption of KOR, which is a langchain extension that allows the extraction and schematization of data manipulated through LLM.

1.4 Thesis Structure

Chap2 - State of Art: This chapter provides a comprehensive review of the current knowledge and developments in the field touched by this work. The focus in on NLP, privacy considerations, Langchain, fine-tuning techniques. This chapter also provides an introduction to ChatGPT ClaudeV2.

Chap3 - Implementation: This chapter describes the datasets used, the architecture of the system, data preparation and storage, data pre-processing, prompt engineering, and the design and functionalities of the chatbot.

Chap4 - Evaluation: This chapter evaluates the results of the project described through data validation and model comparison.

Chap5 - Parallel Project: This chapter describes the parallel project in wich I explored the ability of llms in extraction information process from complex documents, integrating ocr technologies and prompt engineering.

Chap6 - Conclusion & Future Development:This chapter draws the conclusions of the thesis, summarizes the goals achieved and outlines potential future developments in the field.

2. State of the Art

In Sec. 2.1 I will discuss structured and unstructured data, highlighting their differences. In Sec. 2.2, I will trace the temporal evolution of NLP. In Sec. 2.3, I will present synthetic data and various techniques for creating them using NLP methods. Sec. 2.4 will delve into the definition and functioning of Cosine Similarity. In Sec. 2.5 will elucidate LLM models, what fine-tuning is, and the techniques that can be applied to these models. In Sec. 2.6, , I will discuss programming languages, explain technique and software platforms used to move in the thesis domain.

2.1 Data

A *data* is an objective, uninterpreted representation of reality, i.e., an encoded elementary description of a piece of information, entity, phenomenon, transaction, event, or other.[29]

Data can be distinguished into *structured*, *semi-structured*, and *unstructured*.

The *structured data* are those that adhere to a predetermined set of rules, i.e., for which it is possible to define their types and relationships to each other. Structured data depend on a schema and can be represented by rows and columns and stored in a central repository, typically a relational database, from which they can be retrieved separately or in a variety of combinations ready for processing and analysis.

The *semi-structured data* are those that contain semantic tags but do not conform to the structure of relational databases. They are schema-less data, but are represented by labels, graphs, hierarchies and tree structures. These include data in json, xml, etc. format.

The *unstructured data* are characterized by their ”unorganizable” nature, in fact they cannot be represented through a predefined schema. The high organizational complexity of unstructured data means that it is very difficult to manage and process them. These include text, images, video, audio, social-media feeds, etc.

2.1.1 Existing paradigms for processing structured data

Commonly, structured data is handled using RDBMS (Relational Database Management System). The SQL language is frequently employed for interacting with RDBMS systems. Despite the benefits and limitations associated with structured data, it is estimated that only around 25% [7] of the world's data falls into this category, leaving the vast majority of data in the form of unstructured data.

2.1.2 Existing paradigms for processing unstructured data

Unstructured data, as the term implies, lacks a predetermined structure and can manifest in diverse forms and schemas. This adaptability often leads to them being labeled as "free-schema" or "schema-independent," although it's important to clarify that this doesn't imply a complete absence of schema. While unstructured data encompasses qualitative, descriptive, subjective, or non-numerical information, it's inaccurate to solely associate NoSQL databases with unstructured data.

NoSQL, which expands to "not only SQL," is adept at handling structured, semi-structured, and unstructured data alike. MongoDB serves as a prominent example of a NoSQL database. Unstructured data, comprising text, video, and audio, constitutes approximately 80% of the world's existing data.

Contrary to certain misconceptions, unstructured data can still exhibit patterns or contain valuable insights, despite lacking a fixed schema. Data lakes, serving as storage systems, provide a solution for managing diverse data types. They can accommodate unstructured, semi-structured, and structured data from various sources without necessitating a predefined schema, thereby preserving the original data format.

However, it's essential to clarify that the flexibility of data lakes doesn't imply that all data within them is unstructured. Rather, it underscores the data lake's ability to handle heterogeneous data without enforcing a uniform data structure at the outset.

2.1.3 CAP Theorem

In the realm of distributed systems, the CAP theorem stands as a fundamental concept governing the design and behavior of such systems. Proposed by computer scientist Eric Brewer in 2000, the CAP theorem highlights the inherent trade-offs that arise when designing distributed systems concerning Consistency, Availability, and Partition Tolerance [10].

- *Consistency (C)*: Consistency refers to the property that ensures all nodes in a distributed system have the same data at the same time. In other words, every read operation will return the most recent write or an error. Achieving consistency ensures that clients receive accurate and up-to-date information regardless of which node they access.

- *Availability (A)*: Availability indicates that every request receives a response, either success or failure, without any guarantee of the data being the most recent. In simple terms, the system remains operational even if some nodes fail, ensuring continuous service to users.

- *Partition Tolerance (P)*: Partition tolerance refers to the system's ability to continue functioning despite network partitions or communication failures between nodes. In a distributed system, network partitions are inevitable, and ensuring partition tolerance allows the system to remain operational even during these disruptions.

According to the CAP theorem, it is impossible for a distributed system to simultaneously achieve all three properties of Consistency, Availability, and Partition Tolerance. Instead, the theorem asserts that distributed systems can only prioritize two of these properties at the expense of the third. This leads to three possible configurations [1]:

- *CA*: Consistency and Availability are prioritized, meaning that the system always returns the most recent data and remains available for all requests. However, in the event of a network partition, the system might become unavailable.

- *CP*: Consistency and Partition Tolerance take precedence, ensuring that the system remains consistent despite network partitions. However, this might result in temporary unavailability during partitioned states.

- *AP*: Availability and Partition Tolerance are prioritized, allowing the system to continue functioning despite network partitions. However, this might lead to eventual consistency, where different nodes might have slightly different versions of data.

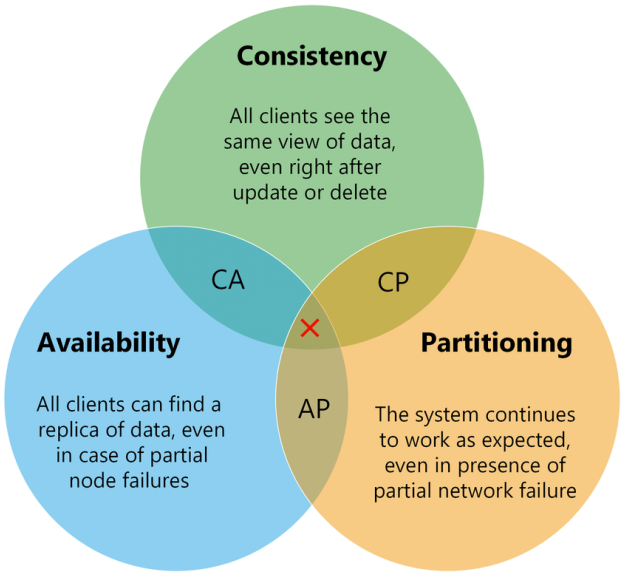


Figura 2.1: CAP Theorem

2.2 History of NLP

The earliest known form of spoken language dates back more than 100,000 years ago, while written language emerged approximately 5,000 years ago. Understanding and analyzing natural language is crucial for extracting information from it, and at its core, Natural Language Processing (NLP) involves teaching machines how to comprehend human language.[51]

The concept of a ”translation machine” first appeared in the early 1930s when Georges Artsrouni attempted to map words from one language to another using a bilingual dictionary and a paper tape. Although this method was rudimentary and did not account for grammar nuances, it represented an initial attempt at automated translation. However, relying solely on conceptual approaches proved unsuccessful in devising a functional method [52].

Consider, for example, how the Rosetta Stone represents a fundamental icon in the history of deciphering Egyptian hieroglyphs and has a profound connection with the field of Natural Language Processing (NLP). Discovered in 1799 during the French occupation of Egypt, this stone

contains a decree issued by King Ptolemy V Epiphanes in 196 BC, inscribed in three different scripts: Egyptian hieroglyphs, demotic, and ancient Greek.

The link between the Rosetta Stone and Natural Language Processing lies in the concept of machine translation and automatic understanding of human language. Just as Champollion deciphered the hieroglyphs, so too do NLP algorithms analyze and interpret text in natural language. Using machine learning models and neural networks, NLP enables machines to understand the meaning of texts, translate them into other languages, generate coherent responses, and much more. Additionally, the Rosetta Stone emphasized the importance of multilingual and multiscriptural data in the process of language understanding. Similarly, in the realm of NLP, processing data from diverse linguistic sources and writing modes is crucial to ensuring accurate and comprehensive analysis of human language.

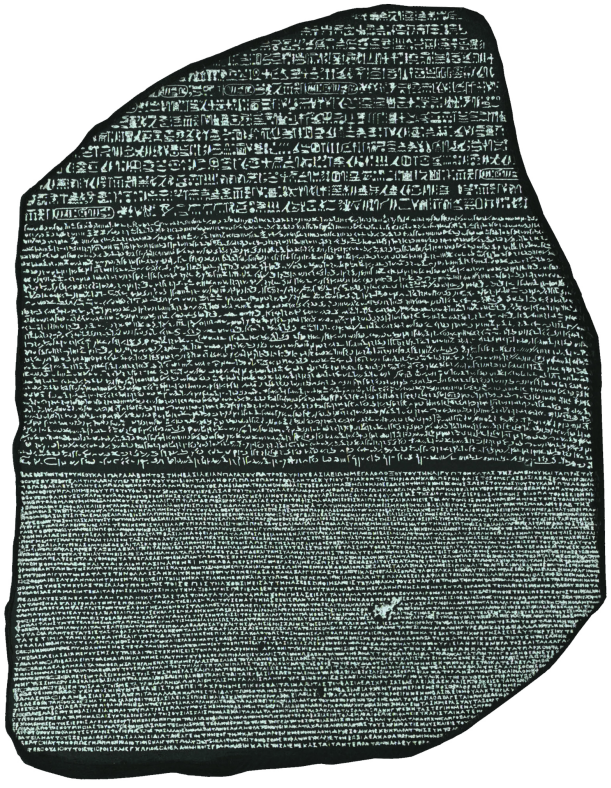


Figura 2.2: Rosetta Stone

2.2.1 Turing Test

The pivotal moment in the evolution of Natural Language Processing (NLP) occurred around 1950, coinciding with the popularity surge of cryptography during World War II [42]. Alan Turing, often hailed as the father of Computer Science and NLP [28], played a central role in this era. For further insight into Turing's contributions, one can explore "Alan Turing: The Enigma" [40] or watch the film "The Imitation Game" [65], both of which delve into his history. These resources detail Turing's utilization of the 'Enigma' machine and his remarkable success in decrypting Nazi messages, a feat that significantly altered the course of World War II. In 1950, Alan Turing proposed the "Turing test" as a means to evaluate whether a computer could exhibit human-like thinking. The test was structured as follows: three participants were involved— a man (player 1), a woman (player 2), and an interrogator. The interrogator's task was to determine the gender of the participants based on their responses to written questions. Player 1 attempted to mislead the interrogator, while player 2 aimed to assist the interrogator in correctly identifying their gender. The test commenced when player 1 was replaced by a machine. If the interrogator successfully identified both genders, the machine failed the test [23].

Noam Chomsky was the first to investigate the role of grammar in machine understanding in 1957, focusing on syntactic structures to establish universal linguistic principles. However, Charles Hockett later critiqued Chomsky's work, asserting that it was based on an overly idealized representation of language.

2.2.2 First Chatbot

Thanks to the Turing test, engineers and researchers embarked on the development of products aimed at simulating conversations between humans as closely as possible. One of the pioneering chatbots in this endeavor is Eliza [54], created by Joseph Weizenbaum, a computer scientist and researcher at MIT (Massachusetts Institute of Technology).

The novelty of Eliza lay in its ability to mimic a conversation akin to that of a psychotherapist, exclusively through written text. While it did not possess true understanding of dialogues, it relied on pattern matching to elicit responses. Despite its limited domain of expertise, many individuals of the time believed they were interacting with a genuine person.

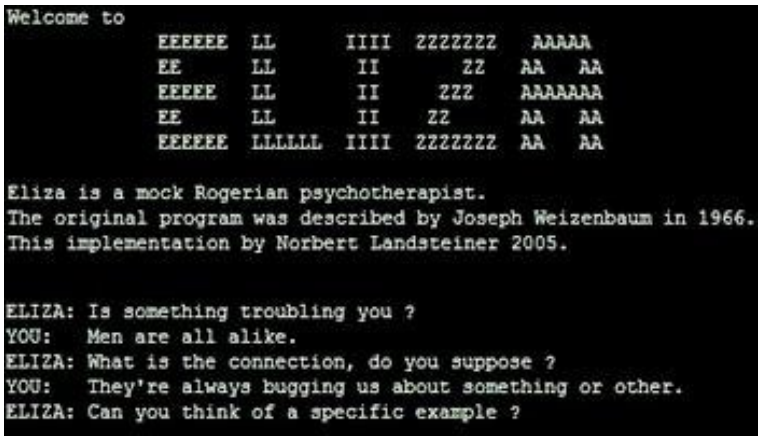


Figura 2.3: Eliza Chatbot

2.2.3 From POS to Machine Learning

One of the major developments in the field of NLP has been the introduction of techniques such as Part-of-Speech Tagging (POS) [32], which involves assigning each word in a text a grammatical part of speech (such as noun, verb, adjective, etc.). This technique, introduced as early as the 1960s and 1970s, has provided a fundamental starting point for many other linguistic analyses, allowing machines to understand the grammatical structure of a sentence and extract deeper meanings from the text [45].

Significant advancements in this domain were catalyzed by the adoption of statistical-probabilistic methods, which paved the way for the development of the initial machine learning models [35] , largely owing to the exponential surge in computational capabilities during the 1980s and 1990s. Machine Learning leverages inferential statistics algorithms such as decision trees, which employ if-then rules to determine optimal outcomes, with probabilistic algorithms providing additional support to these decisions.

Another fundamental technique is Named Entity Recognition (NER) [9], which deals with identifying and classifying relevant entities within a text, such as names of people, places, organizations, dates, etc. This technique has been extensively studied and applied since the 1990s, with the goal of extracting crucial information from large amounts of text and facilitating automatic understanding of documents.

In addition, Information Retrieval (IR) [31] has played a key role in the development of NLP,

enabling machines to retrieve and analyze information from large text corpora efficiently. IR is based on techniques such as indexing, ranking, and information retrieval, which have made it possible to build advanced search engines and handle huge amounts of text data quickly and efficiently.

2.2.4 Deep Learning steps in

A significant advancement occurred in 2010 with the introduction of deep learning and neural networks, addressing many problems that could not be effectively solved with rigid rules and predetermined criteria.

Unlike traditional machine learning approaches, deep learning [26] offers greater versatility, particularly excelling with unstructured data. Unlike previous methods that required converting text into numerical representations while preserving sequence and feature importance through techniques like 'bag-of-words' or 'TF-IDF' (Term Frequency-Inverse Document Frequency), deep learning models such as recurrent neural networks (RNNs) or transformers eliminate the need for such preprocessing. These earlier techniques often resulted in information loss as they failed to capture the contextual importance of words.

2.2.5 Nowdays

With each passing year, NLP techniques are becoming more accessible to individuals who may not possess extensive programming backgrounds. Nowadays, generating code has become significantly easier, facilitated by human language descriptions for tasks like outlining application requirements, creating code templates, and streamlining the debugging process. This transition enables individuals to move from traditional programming languages like R or Python towards embracing low-code or no-code development platforms such as Knime or Orange Data Mining. Furthermore, they can even utilize AI assistants to automate code writing tasks on their behalf.

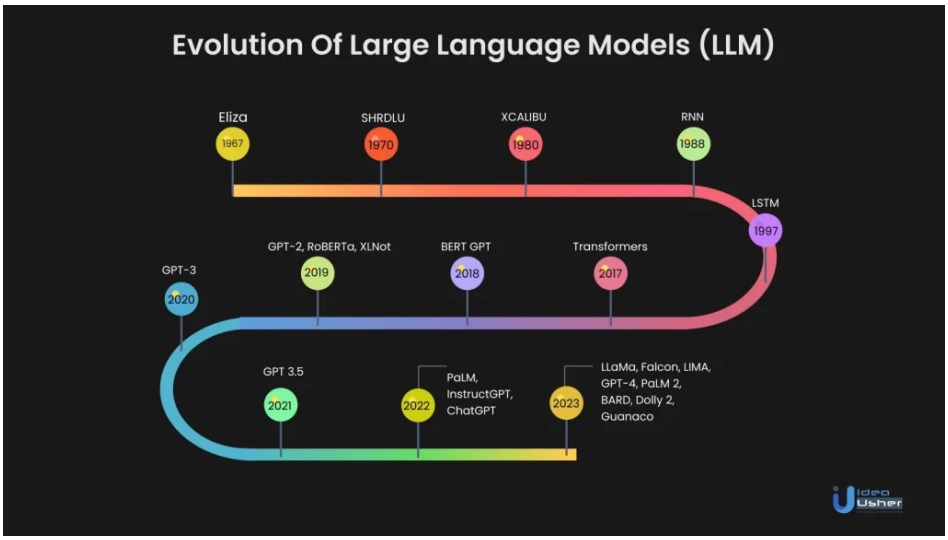


Figura 2.4: Chatbot Timeline

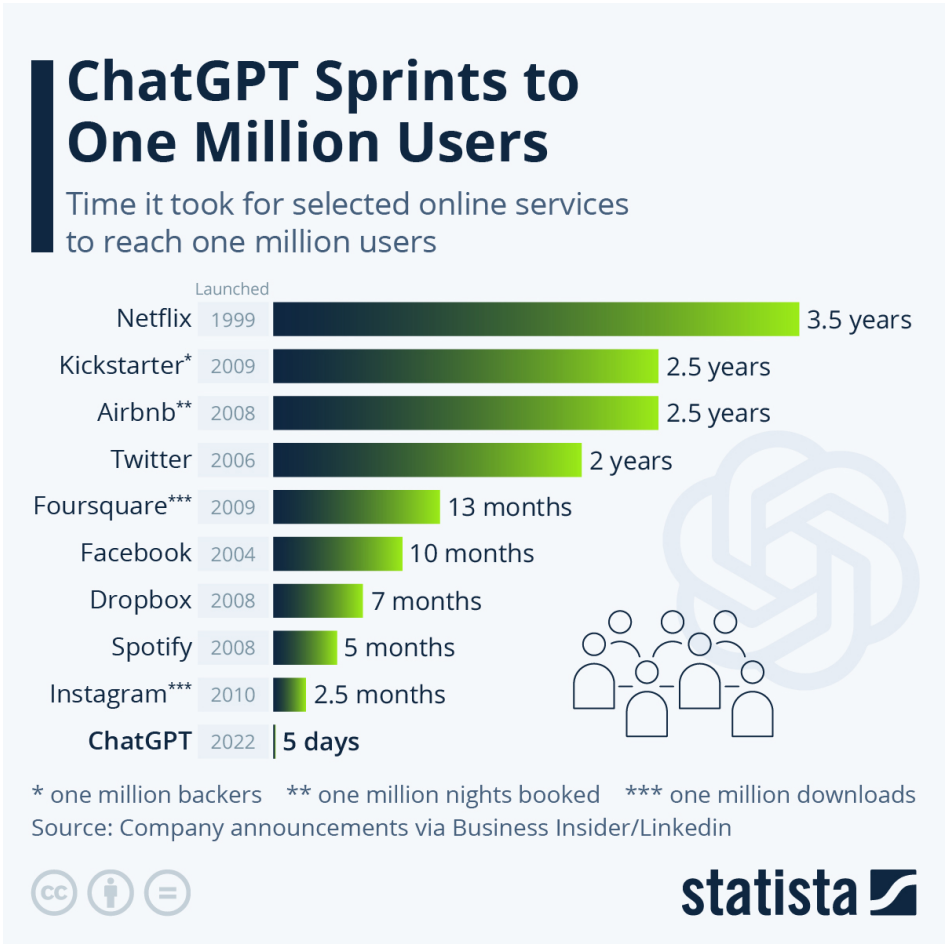


Figura 2.5: 1 Million user

2.3 NLP Techniques for Synthetic Data

Synthetic data refers to information that is artificially generated rather than derived from real-world events. According to Gartner, a leading research and advisory firm, by 2024, we will require 60% less real data to train Machine Learning models [30].

Obtaining real data can often be challenging, consuming significant time and resources, especially when dealing with confidential information such as financial or medical industry data that cannot be accessed or disclosed. In contrast, generating synthetic data that closely resembles real data can result in cost savings, particularly if labeled synthetic data can be produced and utilized for model training. Furthermore, synthetic data enables the use of smaller datasets that may otherwise be overlooked [20].

However, it's important to acknowledge that synthetic data may not always capture the inherent biases present in real-world data, potentially limiting their ability to predict real-life anomalies accurately. Therefore, when opting to use synthetic data, it is essential to conduct a verification phase to assess whether model performance benefits from its inclusion [15].

The quality of synthetic data improves with the refinement of the model, as a better-performing model can generate more reliable synthetic data. Nevertheless, reliance solely on synthetic data is not advisable. It should complement real data, and the integration of both types should be carefully evaluated to ensure optimal model performance and data quality [44].

Numerous NLP techniques exist for generating synthetic data, each applicable to various scenarios. Mostly of these techniques born with Artificial Neural network and include text generation using recurrent neural networks (RNNs), word replacement with similar alternatives facilitated by word embeddings, noise addition, Generative Adversarial Networks (GANs), and Data Masking.

2.3.1 Artificial Neural Networks

The study of *Artificial Neural Networks (ANN)* has been inspired by attempts to emulate biological neural systems. The human brain consists mainly of nerve cells called neurons connected to other neurons by strands of fibers called axons. Axons are used to transmit nerve impulses from one neuron to another whenever the neurons are stimulated. A neuron is connected to the axons of other neurons via dendrites, which are extensions of the neuron's cell body. The point of contact between

a dendrite and an axon is called a synapse. Neurologists have found that the human brain learns by changing the strength of the synaptic connection between neurons upon repeated stimulation from the same impulse. Similar to the structure of the human brain, an ANN is composed of an interconnected assemblage of nodes and direct connections. ANNs are applied to historical data to derive predictions or to make classifications [48].

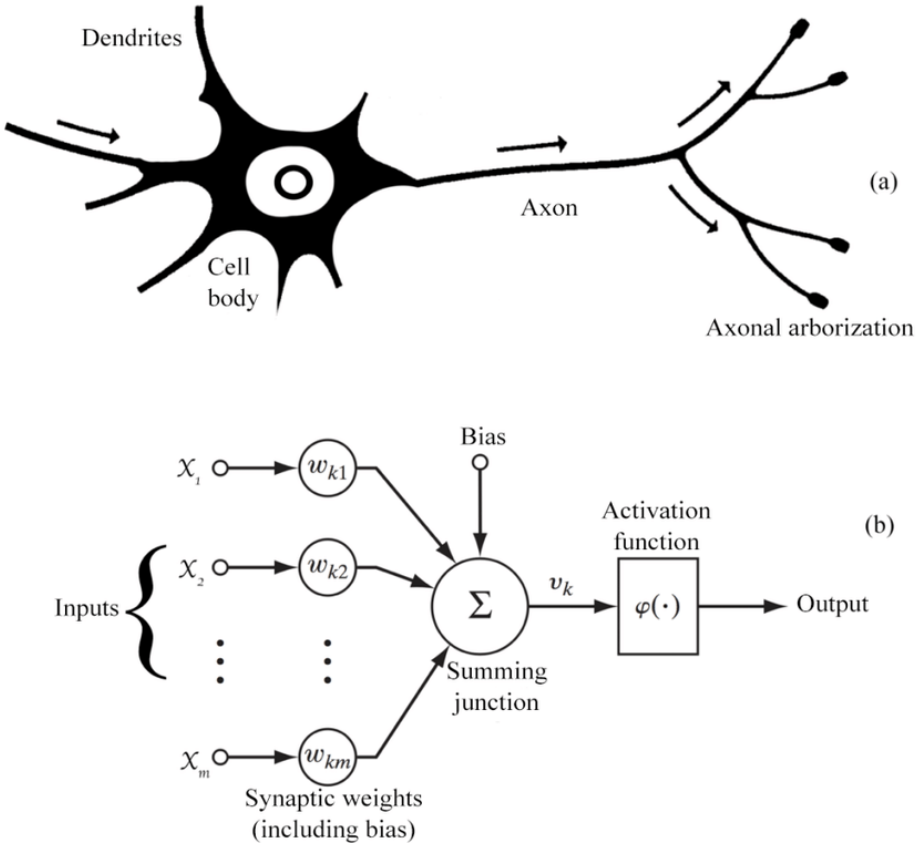


Figura 2.6: Biological neuron vs Artificial neural network

2.3.2 Architecture and Components

Neural Networks consist of layers (layers) of neurons. All neurons contained within a layer are connected to each neuron in the next layer through connections.

Generally, each network is structured as follows:

- *Input Layer*: the input layer associates the active variables in the analysis with the neurons in the hidden layer;

- *Hidden Layer*: the hidden layer is a layer between the input and output layers, in which artificial neurons receive a set of weighted inputs and produce an output through an activation function. Hidden layers can be multiple;

- *Output Layer*: the output layer associates the neurons in the hidden layer with the target variable. The output of the network differs according to the nature of the target variable.

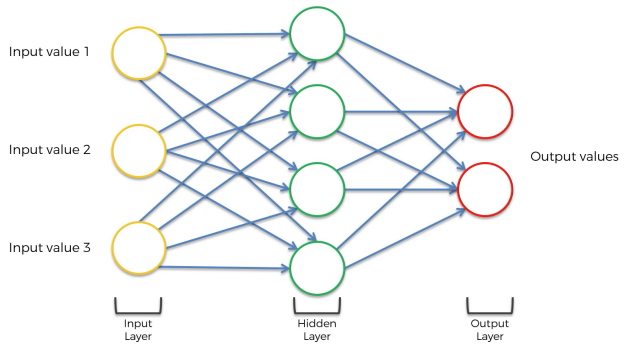


Figura 2.7: Architecture of an Artificial Neural Network with a single hidden layer

2.3.3 RNN and LSTM

The *Recurrent Neural network (RNN)* is a special type of neural network. Unlike traditional ones, the RNN during the processing flow, ”remembers” information encountered in previous processes.This feature makes RNNs optimal in processing sequences of data, such as time series and text data, in which the order in which elements are processed is decisive, as even the elements that come before the one being processed are useful in predicting future elements. [22]

At the heart of the RNN is the *state*, i.e., the memory of the RNN in which everything that has previously occurred during the sequence is stored. This is critical to be able to take all the context into account when making predictions and thus to allow the network to learn long-term dependencies. RNNs are designed by mimicking the way humans process sequences of information, e.g., if we want to know how much it will cost to get gas for the car tomorrow we will have to take into account how much gasoline has cost in the previous days, or if we want to understand a sentence, we take into account the entire sentence and word order and not individual words in bulk.

The architecture of an RNN differs from that of a classical neural network in that there are *loops* within it during processing, which ensure that the information persists and thus is not deleted.

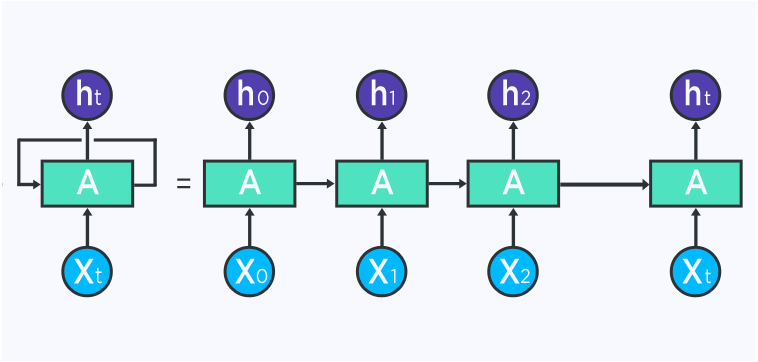


Figura 2.8: Recurrent Neural Network Architecture

Focusing on the details, given the sequence of length $T, X = [x_1, x_2, ..., x_T]$, where x_t denotes the t-th token an RNN uses a recursive mechanism in order to reuse the network result. At step t we would have:

$$o_t, h_t = RNN(x_t, h_{t1})$$

Consider in the figure above the structure on the left side of the equal, A takes input x_t, and returns output h_t, while the arrow that exits A and re-enters A identifies the loop, which allows the information to be remembered, moving from the previous step in the network to the next step. In the right part of the equal, the various steps and thus the individual loops are explicitly represented; in fact, it is shown how the RNN can be viewed as a sequence of traditional neural networks in which each transmits information to the next. This is why the RNN turns out to be the ideal neural network for analyzing sequences of data. However, because of the gradient vanishing or exploding problem, it can be difficult to train RNNs to solve problems that require keeping information in memory for a long time.

A particular type of Recurrent Neural Network is the *Long Short Term Memory Neural Network (LSTM)*. It is the highest performing Recurrent type network and is widely used in Deep Learning. It can be used both for processing single data and for processing sequences of data. LSTM uses memory neurons, to overcome the problem of learning long-term dependencies of RNNs.

The method of information transfer is no different than that of RNNs, however, the information flowing in the network passes through *gates*, which determine what should be stored, what should be forgotten, and what should be returned as output.

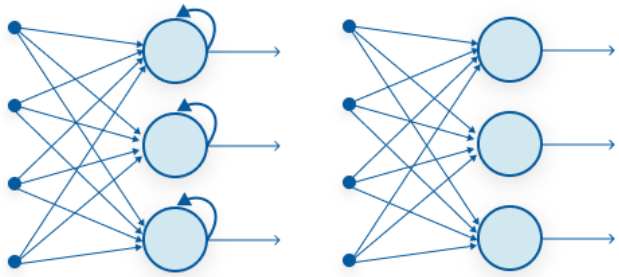


Figura 2.9: Recurrent Neural Network e Forward Neural Network Architectures

2.3.4 Word Embeddings

Word Embeddings represent a foundational modeling technique within the realm of modern Natural Language Processing (NLP). In this approach, words, sentences, and documents are transformed into vectors of real numbers [19], wherein words positioned closely to each other in the vector space are expected to share similar meanings. These models facilitate the computation of word similarity, thereby aiding machines in comprehending the underlying semantics of each word. Consequently, machine learning models leverage these embeddings to enhance performance. For instance, in a numerical space, the word ”Italy” would be closer to ”Rome” than to ”Beijing,” allowing the machine to discern that ”Italy” is more closely associated with ”food” than with ”computer”.[33]

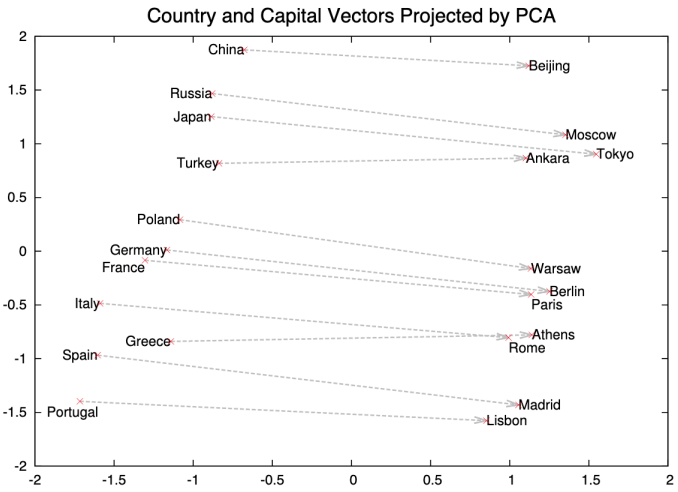


Figura 2.10: Word Embeddings Rapresentation

Word embeddings play a crucial role in Generative Pre-trained Transformers, albeit with a different approach compared to simpler models like Word2Vec or GloVe. In traditional models such as Word2Vec or GloVe, each word is assigned a distinct representation, essentially a ”code” that encapsulatess its meaning. However, GPT models adopt a more intricate methodology. Rather than assigning a solitary ”code” to each word, GPT models analyze the surrounding words to grasp their context. Therefore, when processing a sequence of words, GPT models consider their interrelations, providing a deeper comprehension of the meaning of each word.

2.3.5 GAN

Generative modeling in unsupervised Machine Learning involves discerning patterns within data to replicate them, thereby generating synthetic data resembling real-world data.

GANs (Generative Adversarial Networks) approach this task by employing two opposing neural network submodels: the generator, tasked with generating new instances, and the discriminator, which learns to classify examples as real or fake [18].

These models are trained concurrently, aiming to refine the generator’s outputs until the discriminator can no longer distinguish between real and synthetic data, resulting in a 50% error rate for the discriminator. This signifies the generator’s success in producing convincing, realistic examples.

The appeal of GAN models for synthetic data generation lies in their capability to produce data that closely mirrors reality, facilitated by the adversarial training between the generator and discriminator. While the generator attempts to deceive the discriminator with its output, the discriminator endeavors to differentiate between real and synthetic data. This adversarial process yields high-quality synthetic data.

2.3.6 Data Masking

Data masking is a security measure that involves concealing sensitive information by replacing it with anonymous values. Within the realm of Natural Language Processing (NLP), these techniques are employed to protect textual content containing personal details such as names, addresses, numbers, or financial information [2]. Some of the techniques utilized in this regard include:

- *Masking*: Substituting parts of text with a generic mask, such as replacing ”Michael Jordan” with ”Michael XXXX”.

- *Perturbation*: Introducing noise into sensitive data by anonymizing it, for instance, by adding trivial values to numerical data.

- *Random substitution*:Replacing sensitive data with random values, such as changing a customer’s name to an arbitrary name.

2.3.7 Trasformers

Transformers are currently revolutionizing the field of Natural Language Processing (NLP) and stand as the state-of-the-art for numerous tasks, featuring prominently in various large language models such as BERT , GPT-3 and Claude.[28][17]

Distinguished by their unique approach to deep learning, transformers utilize semi-supervised learning. Initially, they undergo pre-training in an unsupervised manner using extensive unlabeled datasets and subsequently undergo fine-tuning through supervised training to optimize performance.[53][8]

Comprising feed-forward networks and attention blocks, transformers commence with input coding layers and culminate with the Softmax function to generate probabilities. These probabilities are then utilized in NLP tasks to derive the corresponding output for each input word [34].

The architecture of transformers is specifically tailored for Sequence-to-Sequence tasks, where a sequence represents a collection of tokens or, within the realm of NLP, a series of words from a sentence. Crucially, transformers leverage attention mechanisms to discern key elements within a text, enabling them to surpass the limitations encountered by recurrent networks in processing or recalling lengthy texts comprehensively [49].

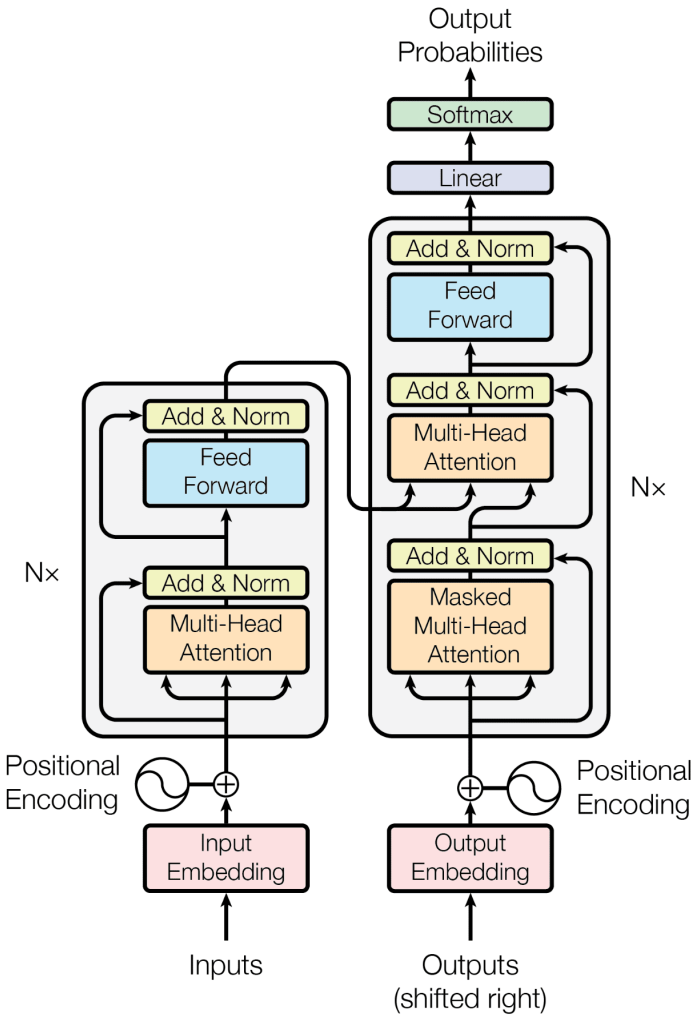


Figura 2.11: Trasformer - Model Architecture

Illustrated in Figure above, the architecture of transformers can be delineated into five distinct

steps.

- *Preprocessing of Input Words*: This stage involves word embedding and positional encoding to prepare the input words.
- *Positional Encoding for Encoding*: This step focuses on positional encoding within the encoders.
- *Preprocessing of Decoded Words*:Similar to the input, this stage includes word embedding and positional encoding for decoded words.
- *Positional Encoding for Decoding*: This step pertains to the positional encoding within the decoder.
- *Output Generation*:The final stage comprises a linear layer followed by a softmax layer for generating the output.

In the initial phase, the input sequence is pre-processed by transforming each word into a vector representation using the embedding technique. To preserve the sequential information of the words, positional encoding, which employs sinusoidal functions, is incorporated into these embeddings.

$$PE_{pos,2i} = sin(\frac{pos}{1000^{\frac{2i}{d_{model}}}})$$
$$PE_{pos,2i+1} = cos(\frac{pos}{1000^{\frac{2i}{d_{model}}}})$$

Where 'i' corresponds to the index of each element and 'd' represents the size of the model, consider the vector [4, 7, 9, 10] as an example:

- Elements 4 and 9 will be coded as the function cosine(i odd)
- Elements 7 and 10 will be coded as the function sine(i even)

The resulting embedded vector and positional encoded vector are merged and forwarded to the first encoder layer. Subsequently, in the Decoded Word Embedding phase, each decoded word from the encoder's output undergoes vectorization using embeddings. This vector serves as the input for the decoder in the subsequent stage. The positional information of the decoded word sequence is

then incorporated into the embedding decoder. Utilizing hidden states, this decoder generates the output sequence token by token, with previously generated vectors serving as context.

Lastly, transformers employ a linear layer followed by a softmax layer [16]. The linear layer transforms the decoder's output vector into a larger logits vector, sized based on the vocabulary learned from training data. The softmax layer then converts these logits into probabilities, with the word having the highest probability being selected as the output.

2.4 Cosine Similarity

Cosine similarity is a metric used to measure the similarity between two vectors, particularly in high-dimensional spaces. In the context of validating outputs from a Language Model (LLM), cosine similarity provides a valuable tool for assessing the semantic similarity between generated text and reference data [13].

2.4.1 Definition of Cosine Similarity

Given two vectors $\mathbf{A}$ and $\mathbf{B}$ in an n-dimensional space, the cosine similarity $\cos(\theta)$ between them is defined as:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

Where $\cdot$ denotes the dot product, and $\|\mathbf{A}\|$ and $\|\mathbf{B}\|$ represent the Euclidean norms of vectors $\mathbf{A}$ and $\mathbf{B}$, respectively.

2.4.2 Applicability in LLM Output Validation

In the context of Language Models, cosine similarity can be employed to compare the similarity between the generated text and ground truth or reference data. By representing the text as vectors in a high-dimensional space, where each dimension corresponds to a unique feature or word, cosine similarity quantifies the semantic similarity between the model's output and the expected outcome.

Cosine similarity offers several advantages for validating the outputs of Language Models[37]:

- **Scale Invariance:** Cosine similarity is invariant to the scale of the vectors, making it suitable for comparing texts of varying lengths.

- **Dimensionality Reduction:** It reduces the dimensionality of the feature space by considering only the angle between vectors, thereby mitigating the curse of dimensionality.

- **Robustness to Noise:** Cosine similarity is robust to noise and irrelevant features, focusing primarily on the semantic content of the text.

2.5 Large Language Models

A Large Language Model (LLM) represents a machine learning model proficient in various Natural Language Processing (NLP) tasks, including text generation, machine translation, and language understanding. These models undergo extensive training on massive text corpora using deep learning algorithms like neural transformer networks, enabling them to discern patterns, linguistic rules, and semantic relationships within text. Consequently, LLMs possess the capability to produce text that closely resembles human-generated content.

The emergence of LLMs has profoundly transformed the landscape of artificial intelligence, offering indispensable solutions for automating tasks reliant on text comprehension and generation. Notable examples of such models include OpenAI's GPT series and Google's BERT [27].

However, alongside their capabilities, LLMs raise ethical concerns. One significant issue pertains to the potential dissemination of misinformation. Given their capacity to generate text indistinguishable from human-written content, there exists a risk of misuse, leading to the creation and propagation of misleading information or fake news, thereby posing considerable challenges in the digital era. Additionally, concerns regarding data security arise since LLMs are trained on vast datasets that may inadvertently contain sensitive or personal information. It is imperative to ensure that these models do not retain or generate outputs based on such sensitive data.

Over recent years, LLMs have gained immense popularity, owing to advancements in deep learning techniques and computational capabilities, such as Graphics Processing Units (GPUs). These advancements have led to a staggering growth in the scale of models, increasing by approximately 5000 times. By 2020, GPT-3 emerged as the model with the highest number of parameters.

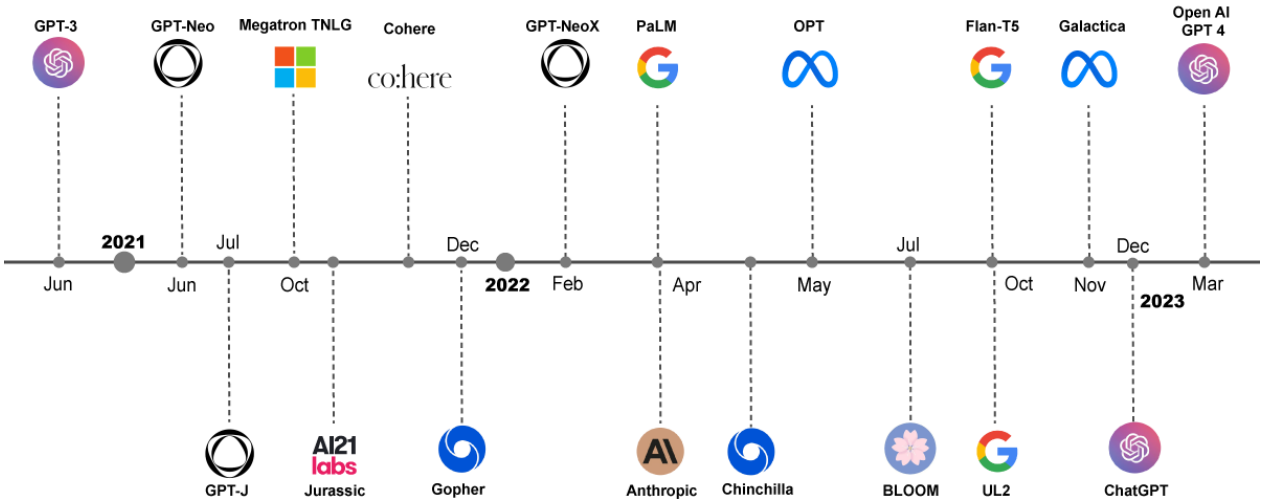


Figura 2.12: LLMs Timeline

2.5.1 Fine-tuning

Fine-tuning in the realm of machine learning refers to a technique aimed at enhancing the performance of a pre-trained model by retraining it on a specific dataset tailored to a particular task. This process involves applying fine-tuning techniques to a training set along with its corresponding labels, during which the model's weights are adjusted based on the new data to acquire detailed insights and optimize performance for the designated task.

This approach proves particularly advantageous when working with limited datasets. By leveraging the knowledge acquired from the pre-trained model, remarkable results can be achieved even with a smaller volume of training data. However, it is crucial to remain vigilant about potential challenges. Biases inherent in the original training data of the pre-trained model may persist post-fine-tuning. Therefore, thorough evaluation of the training dataset used for fine-tuning is imperative. Continuous monitoring of the model's performance throughout the fine-tuning process ensures the delivery of accurate and consistent outcomes [8].

For example, consider the scenario where we have a pre-trained model like GPT, which has been trained on a vast corpus of generic texts. Suppose we aim to utilize this model for sentiment

analysis of movie reviews. In such a case, we can fine-tune GPT using a dataset comprising labeled movie reviews. This fine-tuning operation enables the model to discern pertinent linguistic patterns within the specific domain of movie reviews.[42]

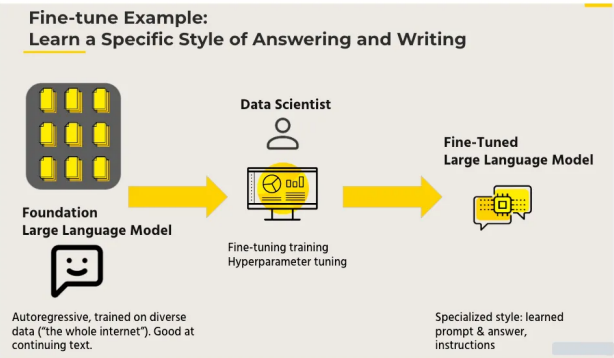


Figura 2.13: LLMs Fine-tuning Process

2.5.2 Fine-tuning Techniques

The fine-tuning techniques for large language models can vary depending on the specific model and the task they are addressing, but they generally include [43]:

- *Transfer Learning*: This technique involves transferring knowledge from a pre-trained model on a large dataset to a specific task through further training on a smaller, more targeted dataset.

- *Learning Rate Scheduling*: Adjusting the learning rate during the fine-tuning process can be crucial for achieving optimal performance. Using dynamic learning rate schedules can help adapt the optimization process more effectively.

- *Regularization Techniques*:Using regularization techniques such as dropout or weight decay can help prevent overfitting during fine-tuning.

- *Data Augmentation*: Adding synthetic or manipulated data to the training dataset can improve the model's ability to generalize better and address data scarcity.

- *Layer Freezing*:This technique involves freezing some layers of the model during the fine-tuning process to preserve the learned features during pre-training and focus training only on certain layers.

- *Task-Specific Head*:Adding task-specific head layers for fine-tuning can help the model focus on the specific task while retaining pre-trained knowledge in lower layers.

These are just some of the common techniques used for fine-tuning large language models, and the choice of techniques often depends on the specific task and characteristics of the available training dataset [46].

2.5.3 GPT - OpenAI

As previously mentioned, GPT stands for Generative Pretrained Transformer, and ”chat” denotes its function as a chatbot designed to interact with humans in a manner as close to real conversation as possible. Released by OpenAI on November 30th, 2022, it experienced exponential growth of $9,900\%$ in just 60 days after launch, attracting approximately 60 million visits per day.

The chatbot is a fine-tuned version of GPT-3.5 (text-DaVinci-003), offered as a free service, while GPT-4 is available with a premium account for $20 per month, providing enhanced computational power and priority in responses, can draw pictures and search on web. Trained using a Reinforcement Learning from Human Feedback (RLHF) approach with datasets comprising approximately 300 billion words (collected from the web, books, and Wikipedia) dating back to 2023. Additionally, the infrastructure is hosted on Microsoft Azure, which has consistently supported and invested in OpenAI, recently reaffirming its commitment with an additional $10 billion in funding [50].

Models of this scale, coupled with instantaneous response times, incur significant computational costs. Tom Goldstein, a professor of Machine Learning at the University of Maryland, has estimated a daily cost of approximately $\$100,000$ and a monthly cost of around $3 million, with the model comprising an estimated 175 million parameters and receiving 10 million queries per day.[41]

2.5.4 Claude V2 - Anthropic

Claude V2[5] is a virtual assistant created by Anthropic, a company specializing in artificial intelligence. It represents an enhanced version of Claude V1, offering increased capabilities and performance. Some features of Claude V2 include:

- It is built on Anthropic's Constitutional AI framework, aimed at making artificial intelligence systems safer, more useful, and honest.
- It has a context window of 200K tokens, allowing it to work with lengthy and complex documents such as source code, financial reports, or literary works.
- It boasts a 50% reduction in hallucination rates, meaning it is less likely to produce false or misleading information. Claude V2 is inclined to admit uncertainty rather than provide incorrect responses.
- It includes a beta feature called "tool use" (like gpt4) enabling it to utilize external tools like calculators, translators, or charts to assist users in solving specific problems or tasks.
- It is accessible via API or through a public beta website, claude.ai or AWS platform, where anyone can register and engage in natural language conversations with Claude, seeking assistance for various activities.

Claude V2 has improved performance, longer responses, and can be accessed via API as well as a new public-facing beta website, claude.ai. We have heard from our users that Claude is easy to converse with, clearly explains its thinking, is less likely to produce harmful outputs, and has a longer memory. We have made improvements from our previous models on coding, math, and reasoning. For example, latest model scored 76.5% on the multiple choice section of the Bar exam, up from 73.0% with Claude 1.3. When compared to college students applying to graduate school, Claude 2 scores above the 90th percentile on the GRE reading and writing exams, and similarly to the median applicant on quantitative reasoning.

2.6 Technologies, Ecosystems and Programming Language

2.6.1 Prompt Engineering

Prompt engineering is a structured and strategic approach aimed at guiding the behavior and output of generative language models, such as ChatGPT. It is a crucial process in interacting with such models, as it allows for influencing and shaping the responses produced based on the specific needs of users and the goals of the application. Prompt engineering involves designing effective prompts, which are instructions or input texts provided to the model to obtain specific responses or guide text generation [39]. These prompts play a crucial role in determining the model's output, directly influencing the coherence, relevance, and quality of the generated responses. The primary goal of prompt engineering is to obtain responses that meet user expectations and adapt to the specific context of the interaction[12].

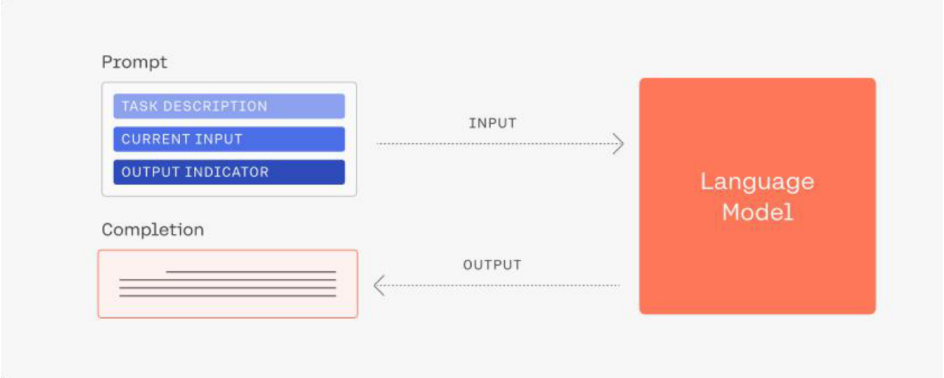


Figura 2.14: Prompt Operation

Prompt engineering involves various key elements that contribute to the design of effective prompts. These include [14]:

- *Clarity and specificity*: The prompt should be clear, specific, and relevant to the question or task at hand. A well-defined structure can avoid ambiguity and guide the model toward a precise response. For example, in the case of automatic translation, a well-structured prompt could include the source and target languages, along with the phrase to be translated.

- *Context*: Providing appropriate context in the prompt is essential for obtaining coherent responses. Providing the model with contextual information can improve the consistency of the responses. For example, if interacting with a chatbot model, the prompt could begin with a summary of the previous dialogue. This allows the model to better understand the interaction context and respond appropriately.

- *Guidance of the type of response*:The prompt should suggest the desired type of response. For example, if a descriptive response is desired, the prompt could begin with ”Provide a description of...”. If a factual response is desired, the prompt could be formulated as a direct question.

- *Request for specific details*: If specific information is desired, the prompt should explicitly request such details. This helps the model focus on what is requested and provide detailed responses.

- *Evaluation of the output*:This technique involves freezing some layers of the model during the fine-tuning process to preserve the learned features during pre-training and focus training only on certain layers.

- *Evaluation of the output*:After obtaining a response from the model, it is essential to evaluate it based on coherence, relevance, and quality. This evaluation can provide valuable feedback for further refining the prompts and achieving better results.

- *Exploration and experimentation*:Prompt engineering often requires exploration and experimentation. Testing different prompt formulations and strategies helps discover which ones lead to the desired responses. Iteration is essential for optimizing prompts over time.

Prompt engineering, however, is not without its challenges. Designing effective prompts may require iterative experiments and adjustments. Language models may also respond unexpectedly due to ambiguity or suboptimal prompt structure. This necessitates a continuous optimization process, where industry professionals constantly evaluate the generated output and adjust the prompts accordingly [11].

Generally, an effective prompt may contain the following elements:

- *Instruction*: defines a specific task or action that the model should perform.
- *Context*: may involve external information or additional contexts that can guide the model towards more accurate responses.
- *Input data*:represents the provided input, such as a question or basic information, for which a response is sought.
- *Output indicators*: specifies the desired type or format for the output generated by the model.

Not all of these components are mandatory in every prompt, as the format and composition depend on the specific task at hand. In the following chapters, more tangible and detailed examples will be presented to further clarify these concepts.

With the power to shape AI output, prompt engineering also raises ethical and responsibility concerns. Designing prompts that could lead to inappropriate or distorted content poses a risk. Therefore, it is essential to consider ethics and ensure that prompts adhere to ethical standards and avoid negative consequences.

2.6.2 Python

Initially conceived by Guido Van Rossum in 1991, Python has since evolved into one of the most prevalent programming languages, celebrated for its user-friendly nature and adaptability across diverse domains including Machine Learning, web development, data analytics, and beyond [40].

Python was deliberately selected as the programming language for this thesis owing to its numerous advantages, particularly in executing transformers. Some of these advantages encompass:

- *Libraries*: Python boasts an extensive array of libraries specifically tailored for machine learning and natural language processing tasks, including TensorFlow, PyTorch, and Hugging Face's Transformers.
- *Strong Community Support*: The Python community, particularly within fields like machine learning and data science, is robust and supportive.
- *Accessibility*:ython is renowned for its simplicity, making it relatively straightforward for individuals to grasp and commence coding compared to its counterparts.

- *Versatility*: Python's flexibility is widely acknowledged and appreciated, allowing developers to adapt it to a myriad of use cases and scenarios.

2.6.3 Amazon SageMaker

Amazon SageMaker is an Amazon platform designed to facilitate the development and deployment of Machine Learning models, with the aim of streamlining the production process for Data Scientists [38].

It provides a range of services that align with the typical pipeline of a Data Scientist, including:

- Data collection and storage.
- Data cleaning and preparation.
- Training and fine-tuning ML models.
- Cloud-based deployment at scale.

These processes are executed using Jupyter notebooks, which support Python or R kernels, and can utilize compute instances tailored to specific requirements. Moreover, users have the flexibility to adjust compute instances throughout different project phases, optimizing costs by scaling resources as needed, such as enabling GPUs or increasing memory.

Additionally, SageMaker offers access to pre-trained models and a suite of monitoring tools, including Amazon CloudWatch, Amazon SageMaker Model Monitor, and Amazon SageMaker Anomaly Detection.

While SageMaker is a robust platform for experienced users familiar with coding, it may not be suitable for those accustomed to drag-and-drop interfaces with minimal coding requirements. In such cases, alternative products like Azure ML Studio are recommended.

2.6.4 Amazon Bedrock

Amazon Bedrock is a fully managed service that provides a selection of high-performance Foundation Models (FM) from leading AI companies such as AI21 Labs, Anthropic, Cohere, Meta, Stability AI, and Amazon, through a single API. It also offers a wide range of features required for

building generative AI applications, using AI safely, securely, and responsibly. With Amazon Bedrock, users can easily experiment and evaluate the best FMs for their use case, privately customize them with their own data using techniques like Optimization and Retrieval Augmented Generation (RAG), and create agents that perform tasks using corporate systems and data sources. Since Amazon Bedrock is serverless, there is no need to manage any infrastructure, and it enables the secure integration and deployment of generative AI functionality into applications using familiar AWS services [4].

- *Amazon Titan*: Text synthesis, generation, classification, open-ended questions and answers, information extraction, embedding, and research.
- *Jurassic-2*: LLM following instructions for any language activity, including answering questions, summarizing, text generation and more.
- *Claude 2*:LLM for thoughtful dialogue, content creation, complex reasoning, creativity and coding, based on constitutional artificial intelligence and harmlessness training.
- *Comanda e incorpora*: Text generation model for business applications and embedding model for searching, clustering or classification in more than 100 languages.
- *Stable Diffusion*: Generating unique, realistic and quality images, artwork, logos and designs.

2.6.5 Langchain

LangChain [25] is a framework for developing applications powered by language models. It enables applications that:

- *Context-aware*: connect a language model to sources of context (prompt instructions, few shot examples, content to ground its response in, etc.).
- *Reason*: rely on a language model to reason (about how to answer based on provided context, what actions to take, etc.).

This framework consists of several parts.

- *LangChain Libraries*: The Python and JavaScript libraries. Contains interfaces and integrations for a myriad of components, a basic run time for combining these components into chains and agents, and off-the-shelf implementations of chains and agents.

- *LangChain Templates*: rely on a language model to reason (about how to answer based on provided context, what actions to take, etc.).A collection of easily deployable reference architectures for a wide variety of tasks.

- *LangServe*: A library for deploying LangChain chains as a REST API.

- *LangSmith*: A developer platform that lets you debug, test, evaluate, and monitor chains built on any LLM framework and seamlessly integrates with LangChain.

Together, these products simplify the entire application lifecycle:

- *Develop*: Write your applications in LangChain/LangChain.js. Hit the ground running using Templates for reference.

- *Productionize*: Use LangSmith to inspect, test and monitor your chains, so that you can constantly improve and deploy with confidence.

- *Deploy*: Turn any chain into an API with LangServe.

The main value props of the LangChain packages are:

- *Components*: composable tools and integrations for working with language models. Components are modular and easy-to-use, whether you are using the rest of the LangChain framework or not.

- *Off-the-shelf chains*: Use LangSmith to inspect, test and monitor your chains, so that you can constantly improve and deploy with confidence.

Off-the-shelf chains make it easy to get started. Components make it easy to customize existing chains and build new ones. The LangChain libraries themselves are made up of several different packages.

- *langchain-core*: Base abstractions and LangChain Expression Language.
- *langchain-community*: Third party integrations.
- *langchain*: Chains, agents, and retrieval strategies that make up an application's cognitive architecture [21].

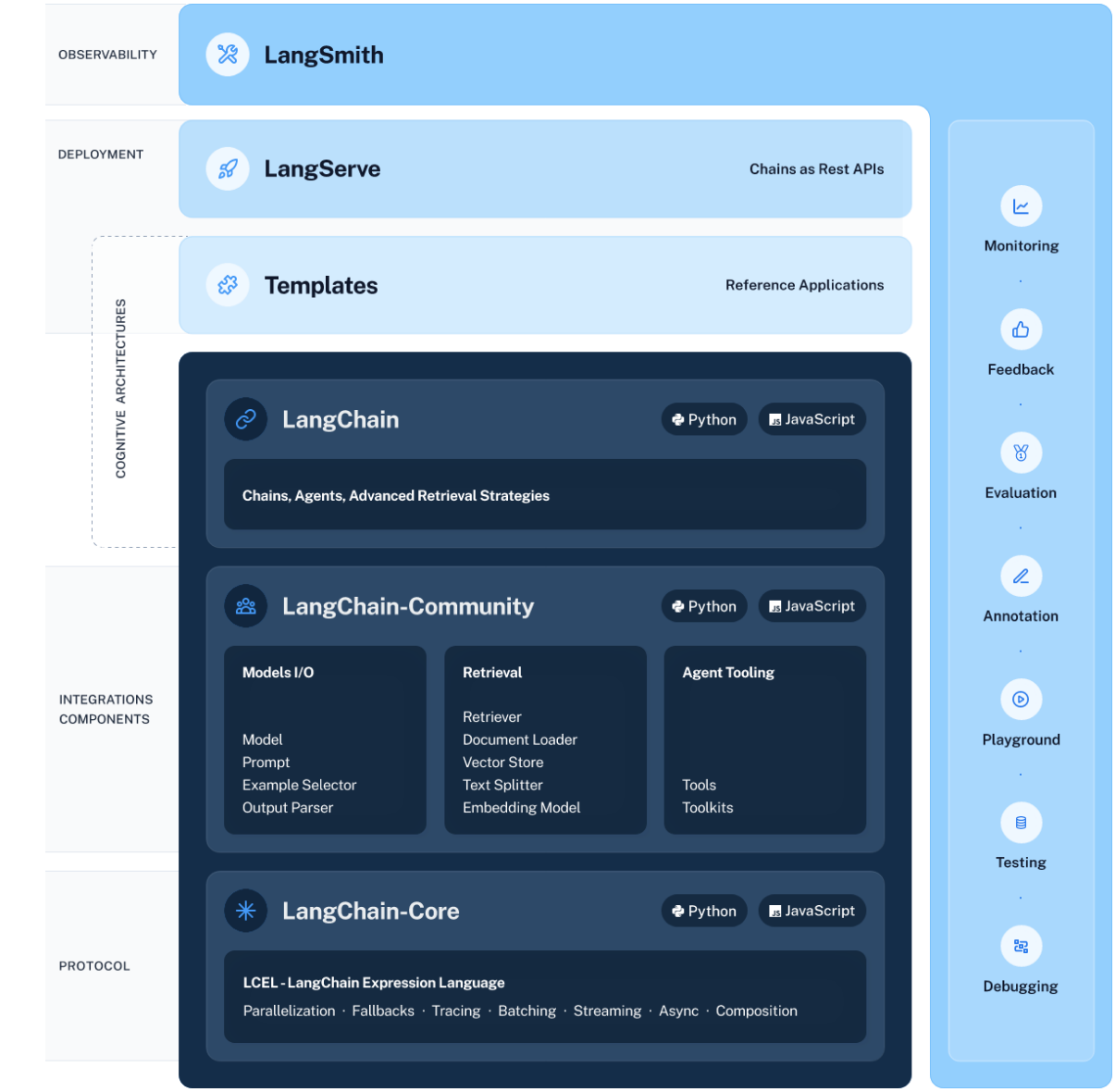


Figura 2.15: Langchain Application Lifecycle

3. Implementation

In Sec. 3.1, 3.2, 3.3 I will describe the alertable tool (architecture and componenets), into which I will have to integrate llm. In Sec. 3.4, I will explain how to configure a data quality rule to date, detailing which parts I should try to synthetize using the LLM. In Sec. 3.5, I describe in detail what kind of data I interface with, how it is processed, how it is structured and how it is stored. In Sec. 3.5 I describe the models used and their characteristics. Sec. 3.7 describe how I use the kor library for data extraction and schematization with llms. In Sec. 3.8 I show an example of synthesization starting from the natural language input and providing as output a configured data quality rule.

The ultimate goal of the project aims at the integration of an LLM within the AlerTable tool, which will be described in the following chapters. The pre-posed goal is to replicate a JSON structure that identifies the configuration of a data quality rule using LLMs. The data quality rule consists mainly of two components Check and Alert (they will be described below in detail). The Model starting from a natural language input and will have to cope with the following challenges: generate the output by ensuring that the data inside the datasets are not disclosed; must be able to test multiple datasources; configure multiple checks that may involve multiple datasources; Launch multiple alerts involving multiple checks.

The selected algorithms that will help achieve the above goals are two: GPT3.5 - turbo and Claude V2.1. The former is an advanced version of the famous ChatGPT, a product pioneered by the OpenAI house and known to anyone since 2021 for paving the way for Generative Artificial Intelligences. The second one, on the other hand, was launched by the company Anthropic, created not surprisingly by former OpenAI employees. The techniques that I will approach alongside these two formidable genAI models, to get to replicate(synthesize) the json structures that define the configuration of data quality rules, will be many. Among these it is necessary to highlight prompt engineering, which is essential to make sure that model responses are consistent and contextualized, and secondly, it is necessary to mention the adoption of KOR, which is a langchain extension that allows the extraction and schematization of data manipulated through LLM.

3.1 Alertable

Any data-driven company must base its decisions on reliable, up-to-date, and high-quality data. A data quality process involves several phases, starting from connecting to the database, applying Data Quality rules, and ending with remediation solutions. These phases often require manual effort, which in some cases could be minimized using an inclusive approach. AlerTable simplifies the Data Quality process by providing an easy-to-use and implement solution. From an end-user perspective, a user-friendly interface simplifies adoption and collaboration across all types of teams, both technical and business. The solution allows transitioning from a rule-based approach scattered among various departments and projects to one guided by a simple and immediate interface, where rules are configured in a few steps, and outcomes can be constantly monitored.

Data freshness, data format, data cardinality, data distribution, data categories, and outlier detection are some of the main checks that can be implemented by defining a check and an alert, hence a Data Quality rule.

The entire computation is performed with Apache Spark, ensuring speed and scalability, and the entire pipeline can be easily deployed in the AWS Cloud environment and accessible, via APIs, to other company tools. The solution has been designed to be easily implemented in an AWS Cloud environment, requiring a pod on Openshift for engine execution, an EMR Spark cluster for rule execution, and a MySQL database for storing metadata and check results.

3.1.1 AlerTable Architecture

The AlerTable architecture, to date, before my contribution needs 5 components:

- Openshift pod containing the docker image of the Java component (both FE and BE);
- Aurora RDS containing the metadata required for AlerTable operation and the outcomes of the various rules executed;
- Secret Openshift where Aurora database references are defined;
- S3 bucket containing details of rule outcomes and logs produced by AlerTable;
- Cluster EMR required for computation of data quality rules.

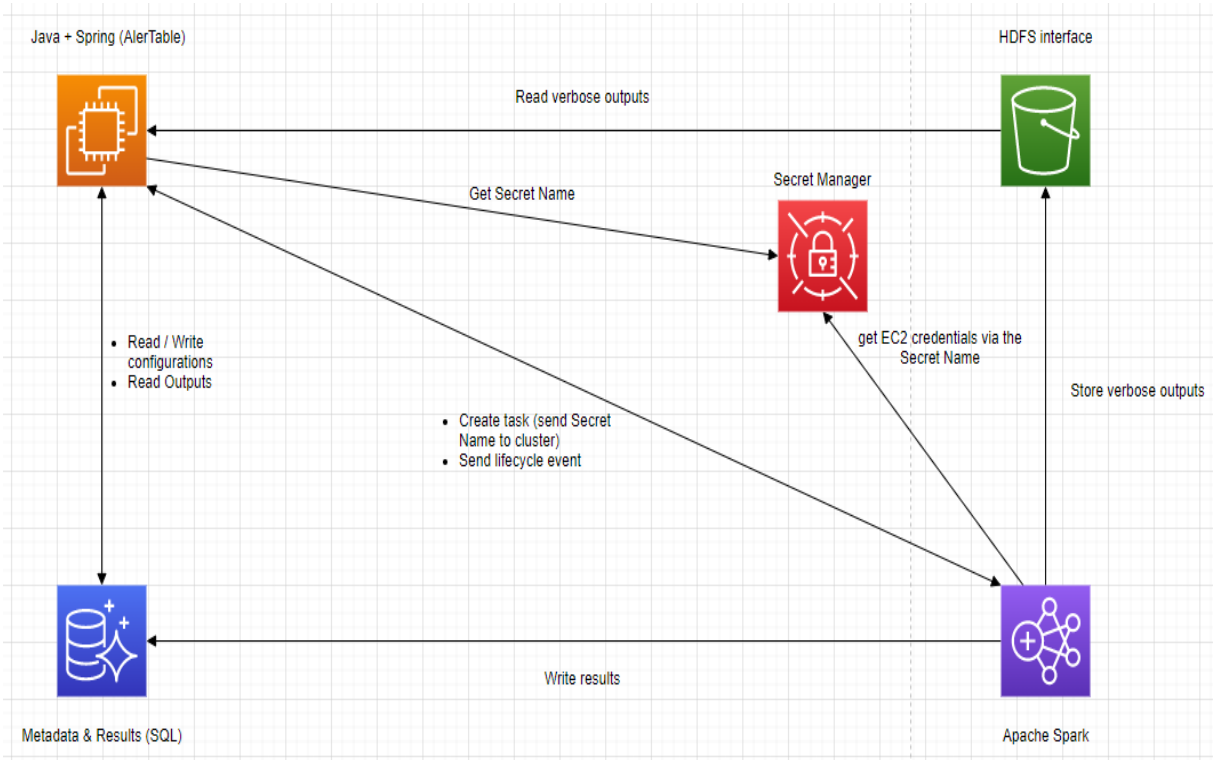


Figura 3.1: AlertTable Architecture

3.2 AlerTable Components

The following is a list of the various components:

- **Datasource**: Ability to configure a specific Datasource starting from a table, view or query present on AWS Athena.
- **Import**: Ability to massively configure multiple Datasources starting from tables, views or queries present on AWS Athena.
- **Checks**: Ability to configure, modify or delete a new/existing data quality check.
- **Alerts**: Ability to configure, edit or a delete a new/existing check.
- **Jobs**. Ability to configure a new job, modify or delete or run an existing job, for each job it is possible to set: which rules are contained in each job, schedule the execution of the job, and define how the outcome of the job run is notified.

3.3 Datasource Import

First, the data sources (data sources) on which data checks need to be performed need to be defined and configured.

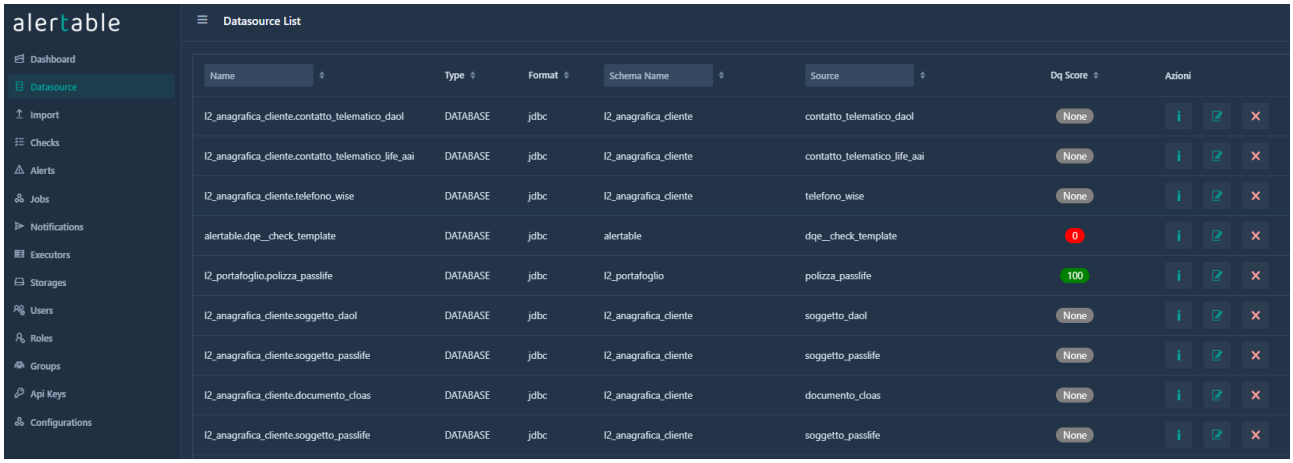


Figura 3.2: List of datasource

In order to proceed with datasource configuration, the Import function can be used. In this way it is possible to create the datasources in a massive way, that is, given a starting database it is possible to configure several databases at once. This function is divided into 2 steps:

- Import Step1 - declaration of database references present on AWS Athena.
- Import Step2 - selection of the tables and properties to be imported

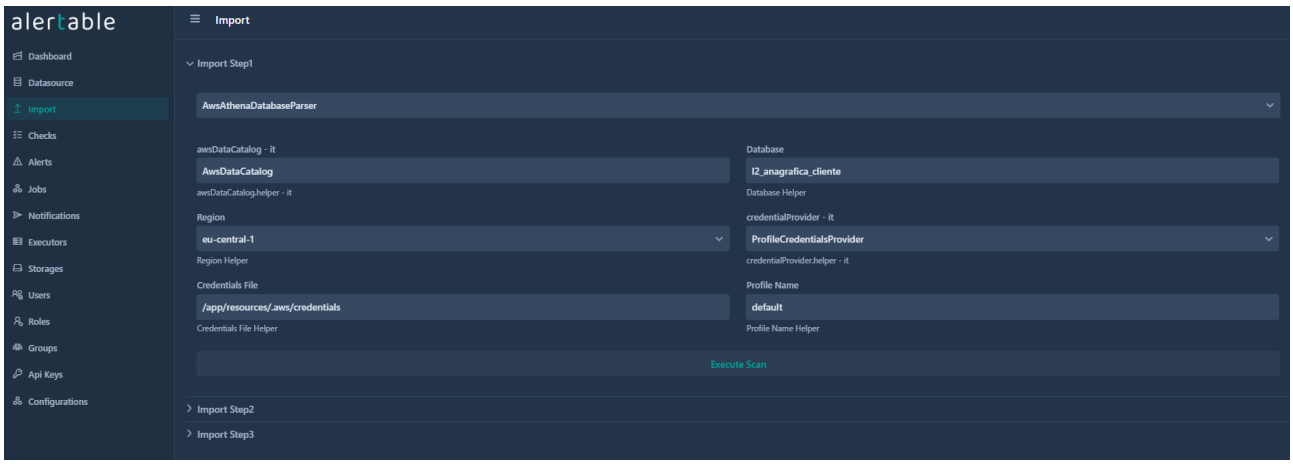


Figura 3.3: Import Function

3.4 Data Quality Rule Configuration

The second step involves configuring the Data Quality rule through two dedicated functions.

- **New Check** - allows the metric of the rule to be defined.
- **New Alert** - allows defining the condition that triggers the notification and the type of notification

3.4.1 Configurazione Check

Below are the input fields required to configure a check:

- **Name** – the displayed name of the check used
- **Description** – textual description of the check
- **Check ID** – the unique identifier of the check
- **Datasource** – identifies the data source on which to apply the check
- **Property** – identifies the column on which to apply the check
- **Check Template** – the type of check it refers to (see below for details)
- **Group By** – multiselect to apply hierarchical groupings
- **Configuration** – dynamic field to enter check hyperparameters such as SQL code
- **External Datasource** – special field to indicate join operations in order to perform checks on cross-table fields
- **Skip Condition** – SQL formatted condition to indicate which rows are not subject to the check

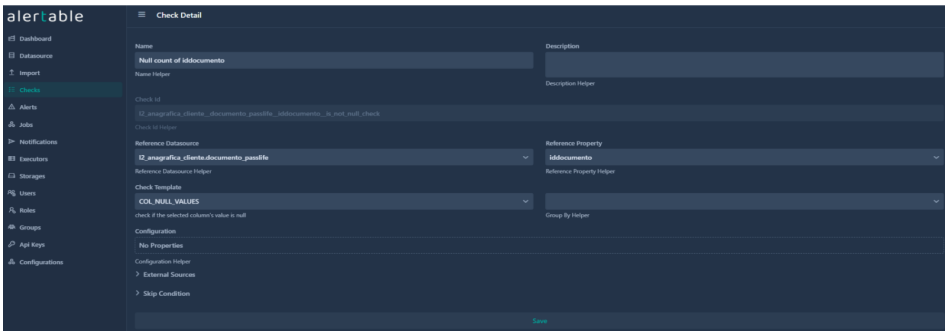


Figura 3.4: Check Configuration

The templates used to configure a check fall into two types: **Check Aggregate** and **Check Detail**. Aggregate Checks produce as output values based on a calculation of an aggregate portion of the data.

Check Name	**Inputs**	**Description**
COL_COUNT	NULL	Returns the count of records.
COL_MAX	NULL	Returns the maximum value taken by the selected column.
COL_MEAN	NULL	Returns the mean of the values taken by the selected column.
COL_MEDIAN	NULL	Returns the median of the values taken by the selected column.
COL_MIN	NULL	Returns the minimum value taken by the selected column.
COL_SUM	NULL	Returns the sum of the values taken by the selected column.
COL_QUANTILE	NULL	Returns the quantile of the values taken by the selected column.
COL_STANDARD_DEVIATION	NULL	Returns the standard deviation of the values of the selected column.
COL_UNIQUE_COUNT	NULL	Returns the count of distinct values of the selected column.

Detail Check produces two different ouputs:

- The first is a single-line true/false flag indicating whether that particular check has failed or not.

- The second is an aggregate summary in which the number of successes, failures, skips, and the success/failure ratio

Check Name	**Inputs**	**Description**
COL_NULL_VALUES	NULL	Check if the value of the selected column is NULL
COL_PAIR_VALUES_A_EQUAL_B	Comparison field	Check if field A is equal to field B
COL_PAIR_VALUES_A_GREATER_B	Comparison field	Check if field A is greater than field B
COL_PAIR_VALUES_A_GREATER_OR _EQUAL_B	Comparison field	Check if field A is greater than or equal to field B
COL_PAIR_VALUES_A_LESS_B	Comparison field	Check if field A is less than field B
COL_PAIR_VALUES_A_LESS_OR _EQUAL_B	Comparison field	Check if field A is less than or equal to field B
COL_VALUES_BETWEEN _BOUNDARIES	lower_bound, upper_bound	Checks if the selected field is within a range of values
COL_VALUES_EQUAL	Comparison value	Checks if the selected field is equal to the comparison value
COL_VALUES_GREATER	Comparison value	Checks if the selected field is greater than the comparison value
COL_VALUES_GREATER _OR_EQUAL	Comparison value	Checks if the selected field is greater than or equal to the comparison value

Check Name	Inputs	Description
COL_VALUES_IN_EXT_SRC	External source	Checks if the values taken from the selected column are present in an external data source
COL_VALUES_IN_SET	Set of values	Checks if the selected field is within a set of values
COL_VALUES_LENGTH _BETWEEN	lower_bound, upper_bound	Checks if the length of the selected field is within a range of values
COL_VALUES_LENGTH_EQUAL	Numeric value representing the comparison length	Checks if the length of the selected field is equal to a comparison value
COL_VALUES_LESS	Comparison value	Checks if the selected field is less than the comparison value
COL_VALUES_LESS_OR_EQUAL	Comparison value	Checks if the selected field is less than or equal to the comparison value
COL_VALUES_REGEX_LIST _MATCH	List of regexes	Checks if the selected field matches at least one of the specified regexes
COL_VALUES_REGEX_MATCH	Regex	Checks if the selected field matches the specified regex
COL_VALUES_TIMESTAMP _FORMAT_MATCH	Timestamp format	Checks if the selected field matches a given timestamp format
COL_VALUES_TYPE_EQUAL	Comparison datatype	Checks if the type of the selected field is equal to the comparison datatype
CUSTOM_QUERY	SQL query	Customized SQL query
IF_THEN_ELSE	Expression identifying IF, ELSE, and THEN	Checks the if_expr. If true, executes the then_expr; otherwise, executes the else_expr

3.4.2 Configurazione Alert

Below are the input fields required to configure an alert.

- **Name**: the displayed name of the alert
- **Description**: textual description of the alert
- **Alert ID**: the unique identifier of the alert
- **Message**: notification message related to the alert
- **Datasource**: identifies the data source on which to apply the alert
- **Property**: identifies the column on which to apply the alert
- **Severity**: weight of the alert in the DQ score calculation
- **Simple Condition**: in case the condition is simple, it can be specified through the no-code interface using the input fields below
- **Complex Condition**: in the field of a complex condition, a TextArea will be displayed where you can write code in the AlerTable meta-language

Below are the fields that define a condition:

Field	Description
Function	Select the function to use for extracting values from the selected checks
Check	Identifier of the check
Field	Type of output of the check to be used
Index Baseline	Temporal filter relative to the time of the data source
Index Run	Temporal filter relative to the time of execution of quality checks
Operator	Operator to use ($>, <, ==, \geq, \leq, !=$)
Term	Comparison type which can be Value or Function. In case of Value, a Term Value field will be displayed where you can enter the value. In case of Function, a form will be presented where you can specify a condition so that the value can be dynamic

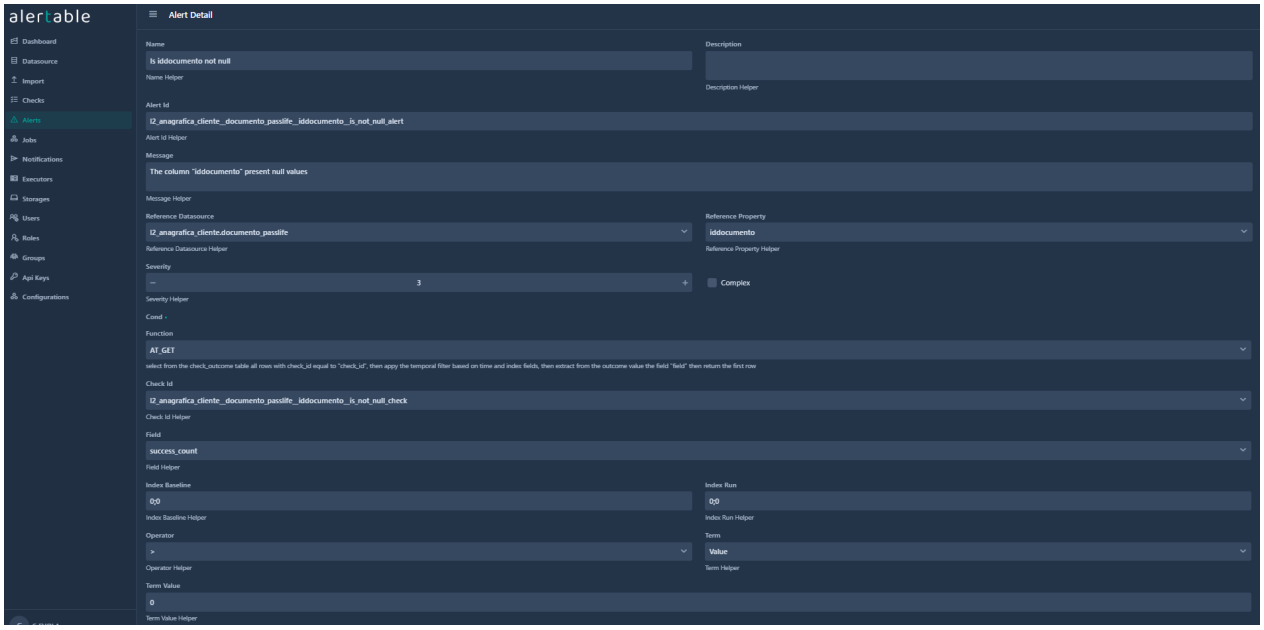


Figura 3.5: Alert Configuration

3.5 Data Preparation and Storage

For each control that is to be implemented, a Json file is generated and then passed to the EMR Spark cluster for rule execution. To date there are 156 Json files available. For each of these there is a Datasource Import, Check and Alert section. Three of these files will be passed to the LLMs as examples in the prompt, obviously the 3 most complex Jsons will be employed so as to account for all possible case scenarios. The remaining files will be used for model validation.

3.5.1 Dataset

Come accennato in precedenza, i dati delle aziende che usufruiscono di questo tool sono immagazinati dentro Aurosa RDS, in formato relazionale. I dataset campione su cui è stata condotto l'analisi sono 9:

- **Anagrafica**: first_name (object), last_name (object), company_name (object), address (object), city (object), county (object), state (object), zip (int64), phone1 (object), phone2 (object), email (object), web (object).
- **Covid**: SNo (int64), Date (object), Country (object), RegionCode (int64), RegionName (object), ProvinceCode (int64), ProvinceName (object), ProvinceAbbreviation (object), Latitude (float64), Longitude (float64), TotalPositiveCases (int64).
- **Survey**: Year (int64), Industry_aggregation_NZSIOC (object), Industry_code_NZSIOC (object), Industry_name_NZSIOC (object), Units (object), Variable_code (object), Variable_name (object), Variable_category (object), Value (object), Industry_code_ANZSIC06 (object).
- **Covid19 Italia**: Direction (object), Year (int64), Date (object), Weekday (object), Country (object), Commodity (object), Transport_Mode (object), Measure (object), Value (int64), Cumulative (float64)
- **Customer Contacts**: interaction_code (int64), masked_customer_code (object), decided_at (object), received_at (object), first_level_group (object), second_level_group (object), action_code (object), action (object), treatment_code (object), treatment (object), channel (object), email (object), outcome (object), propensity (int64), priority (float64),

- **Devices**: deviceId1 (object), deviceId2 (object), timestamp (object), Accelerator_Pedal_Position_1 (float64), Actual_Engine_Percent_Torque (float64), asPre (float64), eFLAP_Actuator_Position (float64), ignition_advance (float64), injection_time (float64), rail_pressure (float64)

- **Geographical_data**: SNo (int64), date_Added (object), Country (object), RegionCode (int64), RegionName (object), ProvinceCode (int64), ProvinceName (object), ProvinceAbbreviation (object), Latitude (float64), Longitude (float64)

- **Report km Auto**: anno (int64), mese (int64), giorno (int64), data (object), targa (object), metri_diurni (int64), id (int64),

- **Sample cs SMS**: REFERENCE_DAY (int64), MSC (object), REPORT_TIME (object), REPORT_COUNT (int64), SMS_TYPE (int64), MSC_TYPE (int64), DX_CAUSE (int64), CLEAR_ADD_INFO_SET_PRB (int64), CLEAR_ADD_INFO_INT_SITUATION (int64), CLEAR_ADD_INFO_INTERNAL_USE (int64), LAC (float64), CELL (float64), IMSI (int64), OPERATOR_CODE (int64), MSISDN_NUMBER_PLAN (int64), MSISDN_NUMBER (float64), IMEI (float64), MS_CODE (float64), SERVICE_CENTER_ADDRESS_NUMBER (float64), ROAMING_STATUS (float64), INCOMING_TS_TIME (object), DELIVERY_TS_TIME (object), SMS_LENGTH (int64), PAGING_TIME (float64), MMS (int64), INTER_PREFIX_MSISDN_NUMBER (float64), INTER_PREFIX_CALLED_NUMBER (float64), INTER_PREFIX_CALLING_NUMBER (float64), INTER_PREFIX_ORIG_CALLING_NUM (float64).

3.5.2 Datasource Import - Json

Below is the first part of the json structure you want to summarize inherent in importing the datasource on which you want to perform the data quality test. The datasources to be imported can be multiple.

```
"datasources": [
    {
        "id": 120,
        "type": "DATABASE",
        "format": "jdbc",
        "path": "devices_daily",
        "columns": ["deviceId1", "deviceId2", "date", "Eng_Percent_Torque", "Eng_Coolant_Temp",
        "Eng_Fuel_Rate", "Eng_Fuel_Temp"],
        "options": { "url": "jdbc:mysql://**********/sample_data", "user": "root", "driver": "com.
        mysql.cj.jdbc.Driver", "dbtable": "sample_data.devices_daily", "password": "**********",
        "pushDownAggregate": "true"},
        "incremental_load": false
    }
],
```


Figura 3.6: Datasource Import Json Structure

3.5.3 Check - Json

Below is the second part of the json structure you want to synthesize inherent in the creation of the data quality check you want to perform. The Checks to be synthesized can be multiple and also can involve multiple datasources.


```
"checks": [
    {
        "id": "check_eng_Percent_Torque",
        "type": "standard",
        "datasource_id": 120,
        "filter_function": "COL_VALUES_GREATER",
        "filter_configuration": {"value": 25.0},
        "aggregation_function": "AGG_COUNT",
        "field_id": "Eng_Percent_Torque",
        "group_by": [],
        "descrizione": "Conteggio dei record che hanno un valore di Eng_Percent_Torque > 25"
    }
],
```


Figura 3.7: Check Json Structure

3.5.4 Alert - Json

Finally, the last part of the json structure to synthetize inherent in alert creation. Alerts can be multiple and also can involve several checks simultaneously.

```
"alerts": [
    {
        "id": "alt_check_eng_Percent_Torque",
        "checks_list": ["check_eng_Percent_Torque"],
        "condition": "AT_GET(\n\tcheck_id=\"check_eng_Percent_Torque\",\n\tfield=\"success_count\",
        \n\tindex_baseline=\";\",\n\tindex_run=\"0;0\"\n) > 0",
        "message": "Il numero di device che hanno una coppia motore (%), registrata ieri,maggiore
        rispetto alla soglia impostata (%)",
        "severity": 1,
        "value": null
    }
]
```


Figura 3.8: Alert Json Structure

3.6 Models

Two separate Large Language Models were chosen to be involved in the implementation of the solution:

- **GPT3.5 - turbo**: via OpenaAi API
- **Claude V2.1**: via AWS Bedrock, Anthropic models

```
llm = ChatOpenAI(
    model_name="gpt-3.5-turbo",
    temperature=0,
    max_tokens=2000,
    openai_api_key=openai_api_key
)
```

Figura 3.9: GPT3.5 - turbo, model parameters

```
llm = BedrockChat(
    model_id="anthropic.claude-v2.1",
    model_kwargs={"temperature": 0.1,
                  "max_tokens": 10000,
                  "top_p": 1}
)
```

Figura 3.10: Claude V2.1, model parameters

Here's a list of adjustable parameters. However, in certain scenarios, the reasoning behind selecting specific values will be explained:

- **Mode**:
  - **Chat**: Designed for multi-turn conversational models. Instead of a single prompt, you can input multiple messages. The model retains the conversation history to generate context-aware responses. Recommended for interactive tasks requiring context awareness.
  - **Complete**: Used to complete a specific prompt, typically for tasks like coding. Provide one or multiple lines of code, and the model completes them.
  - **Edit**: Allows input of text with a specified subsection for the model to modify.
- **Temperature**: A parameter controlling answer variability. Ranges from 0 to 1. Lower values yield more deterministic and focused responses based on the model's training. Higher values result in more diverse and creative responses, albeit less coherent. In this project, deterministic results are preferred.
- **Maximum length**: Limits the total number of tokens (words or parts thereof) to 4096, including both the prompt and output.
- **Top_p**: An alternative to temperature, also known as nucleus sampling. The model considers token probabilities to create a probability mass. Setting it to 0.1 means only tokens representing the top 10% of the probability mass are considered. It's recommended to use either top_p or temperature, but not both simultaneously.
- There are additional parameters, but their discussion is beyond the scope here.

3.7 Kor - Extract Structured Data From Text

For complicated data extraction you need a robust library. The Kor Library [24] (created by Eugene Yurtsev) is an awesome tool just for this. LLMs are great at text output, but they need extra help outputing information in a structure that we want. A common request from developers is to get

JSON data back from our LLMs. In the next two photos we can see how the check configuration of the pattern to be captured by the model takes place. For each json attribute to be configured we have the parameter name and a description of it. We can see for example for the item ”filter configuration” or ”filter function” how a very detailed schema has been passed to the model, this is because there are very precise functions to be defined with associated case histories.

```
check_schema = Object(
    id="checks",
    description="Structured Json",
    # Notice I put multiple fields to pull out different attributes
    attributes=[
        Text(
            id="id",
            description="id della query, costruito con le parole chiave presenti nella query ."
        ),
        Text(
            id="type",
            description="standard"
        ),
        Number(
            id="datasource_id",
            description="id del datasource, {'covid19_italia':198, 'london_merged':204, 'View':139, 'devices_daily':120, 'report_km_auto':184}"
        ),
        Text(
            id="filter_function",
            description="funzione applicata alla colonna di interesse per selezionare le righe d'interesse, la lista delle possibili funzioni è:  ['NO_FILTER', 'CUSTOM_WHERE',
            'COL_VALUES_EQUAL', 'COL_VALUES_GREATER', 'COL_VALUES_GREATER_OR_EQUAL', 'COL_VALUES_LESS', 'COL_VALUES_LESS_OR_EQUAL', 'COL_VALUES_IN_SET',
            'COL_VALUES_IN_EXT_SRC', 'COL_VALUES_BETWEEN', 'COL_VALUES_BETWEEN_LIST', 'COL_VALUES_LENGTH_EQUAL', 'COL_VALUES_LENGTH_BETWEEN', 'COL_VALUES_TYPE_EQUAL',
            'COL_VALUES_REGEX_MATCH', 'COL_VALUES_REGEX_MATCH_LIST', 'COL_VALUES_TIMESTAMP_FORMAT_MATCH', 'COL_PAIR_VALUES_A_EQUAL_B', 'COL_PAIR_VALUES_A_GREATER_B',
            'COL_PAIR_VALUES_A_GREATER_OR_EQUAL_B', 'COL_PAIR_VALUES_A_LESS_B', 'COL_PAIR_VALUES_A_LESS_OR_EQUAL_B'] "
        ),
        Text(
            id="filter_configuration",
            description="dizionario composto come {'chiave': 'valore per filtrare'}, di seguito tutte le casistiche del dizionario date le filter function:
            COL_VALUES_TYPE_EQUAL: {'type':'string'}, COL_VALUES_IN_SET: : {'input_set': array}, COL_VALUES_IN_EXT_SRC: {'external_source': 'string', 'external_source_col':
            'string'}, COL_VALUES_TIMESTAMP_FORMAT_MATCH: {'timestamp_format': 'string'}, CUSTOM_WHERE: {'where_expr': 'sql'}, COL_VALUES_EQUAL: {'value': 'string'},
            COL_VALUES_GREATER: {'value': number}, COL_VALUES_GREATER_OR_EQUAL: {'value': number}, COL_VALUES_LESS: {'value': number}, COL_VALUES_LESS_OR_EQUAL: {'value':
            number}, COL_VALUES_LENGTH_EQUAL: {'len': number}, COL_VALUES_LENGTH_BETWEEN: {'lower_bound': number, 'upper_bound':number}, COL_VALUES_REGEX_MATCH:
            {'regex':'string'}, COL_PAIR_VALUES_A_EQUAL_B: {'col_b': 'string'}, COL_PAIR_VALUES_A_GREATER_B: {'col_b': 'string'}, COL_PAIR_VALUES_A_GREATER_OR_EQUAL_B:
            {'col_b': 'string'}, COL_PAIR_VALUES_A_LESS_B: {'col_b': 'string'}, COL_PAIR_VALUES_A_LESS_OR_EQUAL_B: {'col_b': 'string'}, NO_FILTER: {}, AGG_COUNT_IF:
            {'expr':'string'}, AGG_COUNT_IF_RATIO: {'expr':'string'}, AGG_COUNT_NULL_VALUES: {}, AGG_COUNT: {}, AGG_SUM: {}, AGG_MAX: {}, AGG_MIN: {}, AGG_MEAN: {}, AGG_MEDIAN:
            {}, AGG_QUANTILE: {}, AGG_STDDEV: {}, AGG_COUNT_DUPLICATES: {}, AGG_COUNT_UNIQUE_VALUES: {}, COL_VALUES_BETWEEN: {'lower_bound': number, 'upper_bound': number},
            COL_VALUES_REGEX_MATCH_LIST: {'regex_list': array}"
        ),
```

Figura 3.11: Check Prompt - Kor Schema part1

```
    Text(
        id="aggregation_function",
        description="funzione di aggregazione applicata alla colonno di interesse, la lista delle possibili funzioni è: ['AGG_COUNT',
        'AGG_SUM', 'AGG_MAX', 'AGG_MIN', 'AGG_MEAN', 'AGG_MEDIAN', 'AGG_QUANTILE', 'AGG_STDDEV', 'AGG_COUNT_DUPLICATES',
        'AGG_COUNT_UNIQUE_VALUES', 'AGG_COUNT_NULL_VALUES', 'AGG_COUNT_IF', 'AGG_COUNT_IF_RATIO']"
    ),
    Text(
        id="field_id",
        description="il nome della colonna a cui applicare le funzioni precedenti"
    ),
    Text(
        id="group_by",
        description="lista delle colonne in formato string per cui raggruppare"
    ),
    Text(
        id="descrizione",
        description="query di input in linguaggio naturale"
    )
],
many=True,
```

Figura 3.12: Check Prompt - Kor Schema part2

In the picture below we can see two examples of check configuration passed to the model

```
examples=[
    (
        "Conteggio applicato al campo id del datasource report_km_auto.",
        [
            {
                "id": "Count_car_km_report_records",
                "type": "standard",
                "datasource_id": 184,
                "filter_function": "NO_FILTER",
                "filter_configuration": {},
                "aggregation_function": "AGG_COUNT",
                "field_id": "id",
                "group_by": [],
                "descrizione": "Conteggio applicato al campo id del datasource report_km_auto"
            },
        ],
    ),
    (
        "Conteggio dei record che hanno un valore di cnt > 100 del datasource london_merged.",
        [
            {
                "id": "ck_count_cnt_greater_than_100",
                "type": "standard",
                "datasource_id": 204,
                "filter_function": "COL_VALUES_GREATER",
                "filter_configuration": {
                    "value": 100.0
                },
                "aggregation_function": "AGG_COUNT",
                "field_id": "cnt",
                "group_by": [],
                "descrizione": "Conteggio dei record che hanno un valore di cnt > 100 del datasource london_merged"
            },
        ],
    ),
```

Figura 3.13: Check Prompt - Kor Schema examples

In the next two photos as well as for the check we can see how the configuration of the Alert pattern that the model is to capture takes place. For each json attribute to be configured again we have the parameter name and a description of it. We can see for example how the most complex parameter and one that required multiple configurations is ”condition” this is because in case of a dynamic condition it becomes very complex for the model to capture the json structure for the configuration.

```
alert_schema = Object(
    id="alert",
    description="Structured Json for data quality alert, se l'alert ha successo c'è un problema di data quality",

    # Notice I put multiple fields to pull out different attributes
    attributes=[
        Text(
            id="id",
            description="il nome visualizzato dell'alert, costruito con le parole chiave presenti nell'input o scritto dall'utente."
        ),
       # Text(
       #     id="description",
       #     description="Query di input in linguaggio naturale"
       # ),
        Text(
            id="checks_list",
            description="lista dei check dati dall'utente"
        ),
        Text(
            id="condition",
            description="condizione dell'alert definita dall'utente relativa al check selezionato. Dove le possibili function sono: AT_GET, AT_MAX, AT_MIN, AT_MEAN, AT_STD,
            AT_LR, AT_SUM. I possibili field sono: success_count, fail_count, skip_count, null_count, success_ratio, fail_ratio, null_ratio, success_ratio_all, fail_ratio_all,
            skip_ratio_all, null_ratio_all, mean. Gli operatori matematici sono: >, <, >=, <=, ==, !=. index_baseline è il filtro temporale relativo al tempo del data source"
        ),
        Text(
            id="message",
            description="frase di risposta in caso di successo dell'alert condition"
        ),
        Number(
            id="severity",
            description="importanza del controllo"
        ),
        Text(
            id="value",
            description="null"
        )
    ],
```

Figura 3.14: Alert Prompt - Kor Schema part1

```
examples=[
    (
        "Il numero di device che hanno una coppia motore (%), registrata ieri, maggiore rispetto a 0.",
        [
            {
                "id": "alt_check_eng_Percent_Torque",
                "checks_list": ["check_eng_Percent_Torque"],
                "condition": "AT_GET(\n\tcheck_id=\'check_eng_Percent_Torque\',\n\tfield=\'success_count\',\n\tindex_baseline=\';\',
                \n\tindex_run=\'0;0\'\n) > 0",
                "message": "Il numero di device che hanno una coppia motore (%), registrata ieri, maggiore rispetto alla soglia impostata
                (%)",
                "severity": 1,
                "value": 'null'
            },
        ]
    ),
    (
        "Il conteggio giornaliero è inferiore per più del 80 percento della media nell'ultima settimana, utilizza il check_id
        Count_car_km_report_records",
        [
            {
                "id": "counting_of_car_km_report",
                "checks_list": ["Count_car_km_report_records"],
                "condition": "AT_GET(\n\tcheck_id=\"Count_car_km_report_records\",\n\tfield=\"success_count\",\n\tindex_baseline=\";\",
                \n\tindex_run=\"0;0\"\n) < AT_MEAN(\n\tcheck_id=\"Count_car_km_report_records\",\n\tfield=\"success_count\",
                \n\tindex_baseline=\";\",\n\tindex_run=\"7;1\"\n)*0.8",
                "message": "Il conteggio giornaliero è inferiore per più del 80 percento della media nell'ultima settimana",
                "severity": 1,
                "value": 'null'
            },
        ]
    ),
]
```

Figura 3.15: Alert Prompt - Kor Schema examples

3.8 Input - Prompt

In this section I report figures related to the input that the model receives. It is evident how the new input consisting of the kor schema, the three examples that encapsulate all the case scenarios, and the natural language sentence is a distinct improvement over the forms via a graphical interface that the user would have to fill out to configure a check or alert (see fig 3.4 and 3.5).

This input structure is definitely robust, and thanks to the langchain technology and its kor extension it is possible to create a chain of extraction so that the model remembers the checks created and then employs them in the alert.

```
text="Conteggio dei record che hanno un valore di SNo < 40 del datasource covid19_italia, Conteggio
dei record che hanno un valore di Eng_Percent_Torque > 25 del datasource devices_daily"

chain = create_extraction_chain(llm, check_schema)
output = chain.predict(text=text)['data']

printOutput(output)
```

Figura 3.16: Input Model - Check Example

```
text="Dato il check ck_unipol, rrea un alert che verifica se esistono delle polizze presenti nello storico che non sono presenti nella tabella
online"

chain = create_extraction_chain(llm, alert_schema)
output = chain.predict(text=text)['data']

printOutput(output)
```

Figura 3.17: Input Model - Alert Example

3.9 Output

Thanks to a decidedly robust input structure, which leverages the kor extension as a langchain technology, only three simple configuration examples and a natural language prompt... I report in the figures below the output that both models were able to generate. The check configuration example also shows how the model is able to generate multiple check configurations, on distinct datasets. This is obviously valid for alerts as well.

```
{
   "check": [
      {
         "aggregation_function": "AGG_COUNT",
         "datasource_id": "198",
         "descrizione": "Conteggio dei record che hanno un valore di SNo < 40 del datasource covid19_italia",
         "field_id": "SNo",
         "filter_configuration": "{'value': 40}",
         "filter_function": "COL_VALUES_LESS",
         "group_by": "[]",
         "id": "count_SNo_less_than_40",
         "type": "standard"
      },
      {
         "aggregation_function": "AGG_COUNT",
         "datasource_id": "120",
         "descrizione": "Conteggio dei record che hanno un valore di Eng_Percent_Torque > 25 del datasource devices_daily",
         "field_id": "Eng_Percent_Torque",
         "filter_configuration": "{'value': 25}",
         "filter_function": "COL_VALUES_GREATER",
         "group_by": "[]",
         "id": "count_Eng_Percent_Torque_greater_than_25",
         "type": "standard"
      }
   ]
}
```


Figura 3.18: Output Model - Check Example


```
{
   "alert": [
      {
         "checks_list": "['ck_unipol']",
         "condition": "AT_GET(\n\tcheck_id='ck_unipol',\n\tfield='fail_count',\n\tindex_baseline=';',\n\tindex_run='0;0'\n) > 0",
         "description": "",
         "id": "check_unipol",
         "message": "Esistono delle polizze presenti nello storico che non sono presenti nella tabella online",
         "severity": "1",
         "value": "null"
      }
   ]
}
```


Figura 3.19: Output Model - Alert Example

It can be seen that the three input components were a winning recipe for synthesizing the json file for configuring data quality rules. And how models without ever having seen sensitive customer data manage to capture the semantics of the natural language input and generate the structured data needed for quality verification.

4. Evaluation

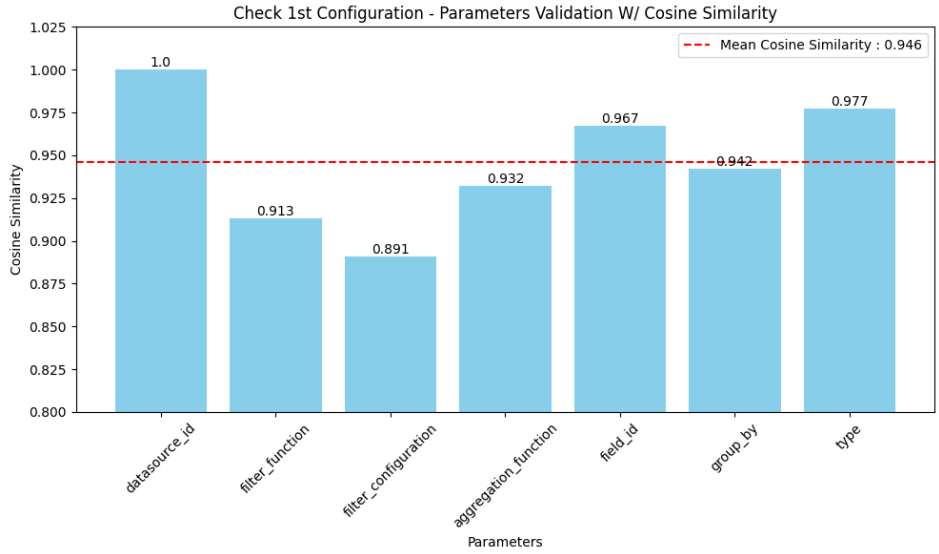
This chapter outlines the results attained during the previous four-month period, pull over them with outcomes generated by the OpenAI GPT3.5 - turbo and Anthropic Claude V2.1. Such a comparative analysis facilitates the derivation of conclusions across multiple criteria. For data evaluation, we opted to decompose the Check and Alert components and calculate the cosine similarity between the structured output generated by the model and the available JSON files that the model has never encountered before.

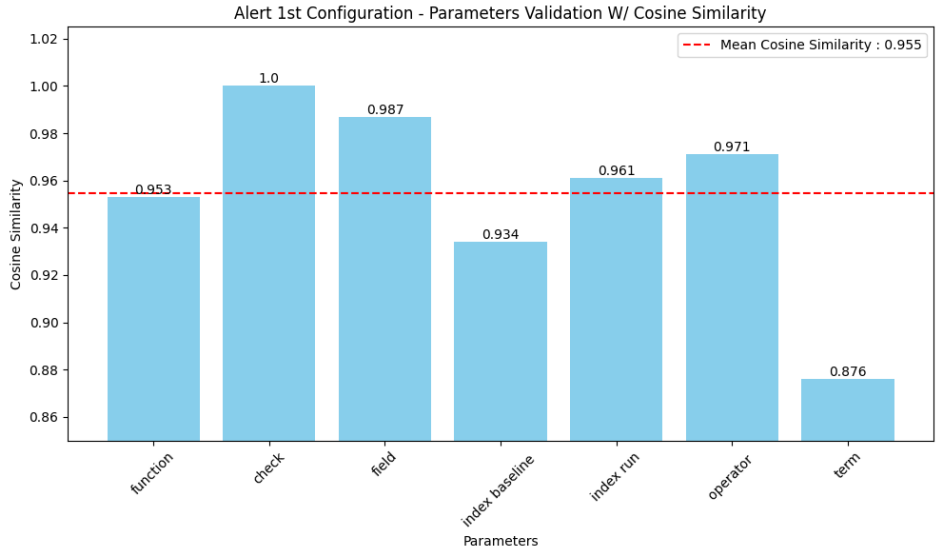
4.1 Synthetic Data Validation

In the following images we can see the cosine similarity at 1st step beetween the check parameters generated by the model and the check parameters from the ref 156 JSon files.

The graph illustrates parameter validation in the first configuration of a check, using cosine similarity as a metric. On the x-axis are the parameters used during the check configuration. On the y-axis is cosine similarity, which measures the similarity between two data vectors. Each bar on the graph represents the similarity for a given parameter. The similarity values vary between 0.9 and 1.

The average cosine similarity is about 0.953, indicating good overall consistency between the configured parameter values and the reference values. The similarity varies slightly among the different parameters, but generally remains high and close to 1, suggesting effective validation of the parameters during check configuration. In conclusion, the graph provides a detailed overview of cosine similarity for each parameter in the first check configuration, highlighting effective validation of the parameters against the reference data.





In the following table we can see the comparision between models of the iterative process results for check schema and input configuration.

To compare the two tables, it is necessary to consider that the cells in the tables represent the cosine similarity between different check configurations. The higher the similarity value, the greater the similarity between the check configuration and the reference data. Below, I will describe the results in detail.

The Claude V2.1 model shows good similarity with the reference data for all check configurations. All but one parameter has a similarity of at least 0.9 with the reference data for each configuration. The parameter ”Type” shows perfect similarity (1) for all check configurations.

Tabella 4.1: Check Cosine Similarity W/ Claude V2.1

	Conf1	**Conf2**	**Conf3**	**Conf4**	**Conf5**
Datasource ID	1	1	1	1	1
Filter Function	0.913	0.941	0.955	0.981	1
Filter Configuration	0.891	0.896	0.935	0.979	1
Aggregation Function	0.932	0.975	1	1	1
FieldID	0.967	1	1	1	1
GroupBy	0.942	0.987	1	1	1
Type	0.977	0.986	1	1	1

The GPT3.5-turbo model also shows good similarity with the reference data, but with some differences from model 1. The similarity of ”Filter Function” and ”Filter Configuration” parameters is lower than model 1 for some check configurations. However, the similarity of the parameters ”Aggregation Function”, ”FieldID”, ”GroupBy”, and ”Type” is perfect (1) for all check configurations.

Tabella 4.2: Check Cosine Similarity W/ GPT3.5-turbo

	Conf1	**Conf2**	**Conf3**	**Conf4**	**Conf5**
Datasource ID	1	1	1	1	1
Filter Function	0.951	0.971	1	1	1
Filter Configuration	0.934	0.953	0.989	1	1
Aggregation Function	0.962	0.986	0.957	1	1
FieldID	1	1	1	1	1
GroupBy	1	1	1	1	1
Type	1	1	1	1	1

By making a comparison between the two LLMs the following can be stated. Both models show good similarity with the reference data for most of the check configurations. The Claude V2.1 model tends to have slightly higher similarity for the parameters ”Filter Function” and ”Filter Configuration,” while the GPT3.5-turbo model shows higher similarity for the parameters ”Aggregation Function,” ”FieldID,” ”GroupBy,” and ”Type.” Overall, both models demonstrate acceptable accuracy in check configuration, although there are some variations in the similarities of individual parameters between the two models. In conclusion, both the Claude V2.1 model and the GPT3.5-turbo model show a perfect level of cosine similarity with reference data in check configuration.

In the following table we can see the comparision between models of iterative process results for alert schema and input configuration. The parameters ”term” include the dynamic condition too.

The Claude V2.1 model shows good similarity with reference data for most alert configurations. All parameters, with the exception of ”Term,” have more than 0.9 similarity with reference data for all configurations. The parameters ”Function,” ”Check,” ”Field,” ”Index baseline,” ”Index Run,” and ”Operator” have perfect similarity (1) for some configurations of the alerts. For the ”Term” parameter, multiple configurations had to be used.

Tabella 4.3: Alert Cosine Similarity W/ Claude V2.1

	Conf1	**Conf2**	**Conf3**	**Conf4**	**Conf5**	**Con6**
Function	0.953	0.976	0.982	1	1	1
Check	1	1	1	1	1	1
Field	0.987	1	1	1	1	1
Index baseline	0.932	0.945	0.971	0.987	1	1
Index Run	0.967	1	1	1	1	1
Operator	0.942	1	1	1	1	1
Term	0.876	0.967	0.981	1	1	1

The GPT3.5-turbo model also shows good similarity to the reference data for most of the alert configurations. All parameters, except ”Term,” have more than 0.9 similarity to the baseline data

for all configurations. The parameters ”Function,” ”Check,” ”Field,” ”Index baseline,” ”Index Run,” and ”Operator” have perfect similarity (1) on the latest alert configurations.

Tabella 4.4: Alert Cosine Similarity W/ GPT3.5-turbo

	Conf1	**Conf2**	**Conf3**	**Conf4**	**Conf5**	**Con6**
Function	0.962	0.996	0.987	1	1	1
Check	1	1	1	1	1	1
Field	0.988	1	1	1	1	1
Index baseline	0.944	0.965	0.991	1	1	1
Index Run	0.983	1	1	1	1	1
Operator	0.963	1	1	1	1	1
Term	0.856	0.884	0.897	0.941	0.981	1

In conclusion, we can say that both LLMs demonstrate high similarity with reference data for most alert configurations. Both models obtain perfect similarities from the start for key parameters such as ”Function”, ”Check”, ”Field”, ”Index baseline”, ”Index Run”, and ”Operator” in different configurations of the alerts. The Claude V2.1 model seems to capture more than the GPT3.5-turbo model the ”Term” parameter in some alert configurations, especially for dynamic case histories. Overall, both models demonstrate perfect accuracy in alert configurations, with very similar results between them.

4.2 Performance Comparison

The graph below illustrates the trend of the average cosine similarity across the steps of control configuration (check), comparing two different timelines: ”GPT3.5 - turbo” and ”Claude V2.1.” On the x-axis are the steps of check configuration, numbered from 1 to 5. On the y-axis is shown the average cosine similarity. This value represents how similar the synthetically generated data are to the actual reference data.

In both timelines, the cosine similarity gradually increases as one proceeds through the control setup steps, from initial values of approximately 0.946 and 0.978, respectively. The ”GPT3.5 - turbo” timeline shows a slightly faster trend in increasing similarity than ”Claude V2.1.”

Both timelines reach a maximum similarity peak of 1, indicating that the synthetically generated data are virtually indistinguishable from the real reference data.

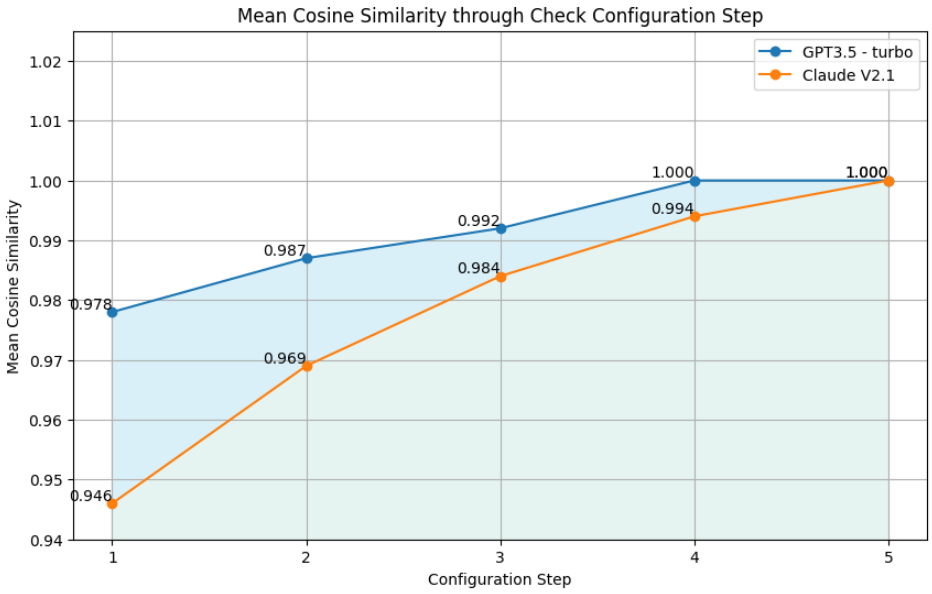


Figura 4.1: Performance Comparison - Check

The graph below highlights the progressive improvement in cosine similarity during the alert configuration using two different synthetic data generation models, confirming the effectiveness of both in producing high-quality synthetic Alerts that are similar to the actual reference data. On the x-axis are the steps of the alert configuration, ranging from 1 to 6. On the y-axis is shown the average cosine similarity. This value indicates how similar the synthetically generated data are to the actual reference data.

In both timelines, the cosine similarity tends to increase as one proceeds through the alert setup stages, from initial values of approximately 0.95 to values of 1, indicating perfect similarity with the reference data. The ”GPT3.5 - turbo” model shows a slightly faster trend in increasing similarity than ”Claude V2.1,” especially in the early stages of pattern setup. Both timelines reach a maximum similarity of 1, indicating that the synthetically generated data are virtually indistinguishable from the actual reference data.

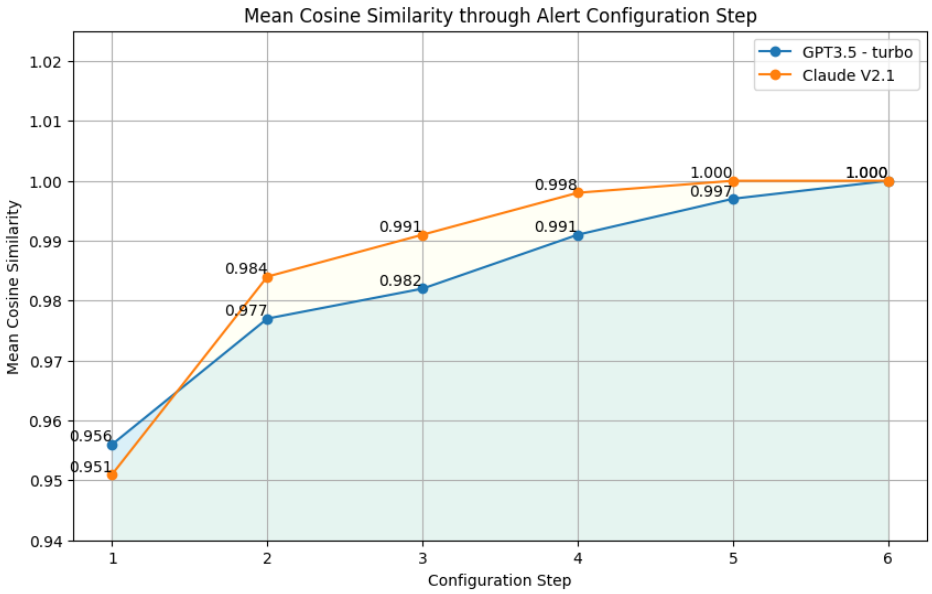


Figura 4.2: Performance Comparison - Alert

There are no significant variations or obvious discrepancies between the two timelines, suggesting that both models produce similar results in terms of cosine similarity during the alert configuration.

Finally, I report the table of costs incurred to implement the project [6] [36]:

Tabella 4.5: Cost Comparison

	GPT3.5 - Turbo	**Claude V2.1**
Open Source	X	X
Cost per Hours	X	$ 86,00 (1 month) or $ 48,00 (6 month)
Cost per Tokens Input	$0.003/1K tokens	$0.008/1K tokens
Cost per Tokens Output	$0.006/1K tokens	$0.024/1K tokens
Max token per call	4,096 tokens	200k tokens
Number of call	$\sim$ 22000	$\sim$ 8500
Total cost	$\sim$ $389	$\sim$ $338

5. Parallel Project

It is proper to write a short chapter about the parallel project I did for the internship part. Obviously, here too, preliminary research was conducted on the state of the art related to the technologies used.

In Sec. 5.1, I will describe the goals and the data of the intership project run in parallel. In Sec. 5.1, I describe in what kind of prepocess was used, cite OCR and segmentation. In Sec. 5.3, 5.4 I describe the models used and the prompt engineering applied to extract information and the post-processing. In Sec. 5.5 I show the results of my intership project.

5.1 Description

The parallel project involved analyzing user manuals and documents for a company specializing in mechanical projects related to transportation. These documents were shared with me in vectorial PDF format. Despite their high quality, I encountered challenges due to my limited knowledge of engines and the complex structure of certain documents.

In this project, the objective was to extract specific information from engine manuals provided by the client using Language Models (LLMs). The client provided five types of PDF files labeled as: CERT, Checklist, ESS, Manual, and ShopOrder. Additionally, a CSV file was provided, detailing the documents and specifying the engine information to be extracted from each. My task was to extract the desired data from the PDFs and validate it against the information provided in the CSV.

Often, the information within the PDFs lacked structure, necessitating the use of LLMs for extraction. The following figure is a representation of a typical page found in these files, provided for illustrative purposes only and not as an original sample due to privacy considerations.

ICON/Gauge	Description	Requirement Level	CAN or wired	CAN Message and signal	Bit value	
	High Coolant Temperature Lamp	Mandatory by FPT	CAN	EDC2BC CAN ID: 18 FF 21 00 Cycle time 50 ms Byte 2 bit 8-6	000	No Warning
					001	Pre-Warning (102°C)
					010	Warning (106°C)
					011	Not defined
					111	Not defined

Table 9 – High engine coolant temperature lamp features

ICON/Gauge	Description	Requirement Level	CAN or wired	CAN Message and signal	Bit value
	Coolant Temperature Meter	Optional	CAN	ET1 CAN ID: 18 FE EE 00 Cycle time 1000 ms Byte 1	1°C / bit

Table 10 – High engine coolant temperature indicator features

Figura 5.1: Page Example

This project utilized AWS services extensively. Initially, all documents slated for analysis were stored in an S3 bucket. Subsequently, Amazon Textract OCR processed these documents, and the results were stored back in the bucket for further processing in the subsequent phase. In this phase, Bedrock and SageMaker were employed. The selected model for this task is Claude Instant, accessible via Bedrock, with a contextual window spanning approximately 9000 tokens.

5.2 Preprocessing

The data initially existed in its raw format. To prepare it for analysis, I utilized Amazon's OCR tool, Textract, to extract and process the data. Following this, I segmented the data to prepare it for input into the model. For pipeline development, I leveraged the Amazon SageMaker software platform. Since the data had already undergone processing within Amazon's suite of services, SageMaker provided an efficient environment for further development and refinement.

5.2.1 OCR

In this scenario, I utilized AWS's Textract OCR algorithm [3] to process the original images, and the outcomes met my expectations. However, I am unable to display the results due to privacy concerns.

Extracting information from these images posed additional challenges due to varying text sizes and inconsistent tabular formatting. The text in the original manuals was often uneven and more blurred compared to the provided illustrative example, making it challenging to decipher. I've included this comparison to highlight the obstacles faced by OCR algorithms.

To address these challenges, I experimented with techniques better suited to the task. This involved segmentation to identify text-containing regions within the images, followed by resizing to magnify each identified segment. Finally, I employed composition techniques to merge all enlarged segments into a new image, ensuring that text did not overlap.

5.2.2 Data Cleaning and Segmentation

The processing of data by a single run from the LLMs can be time-consuming, given the maximum token limit of 9000.

To address this limitation, the text needs to be segmented into chunks, each adhering to a specified threshold. This segmentation process is incorporated into the pre-processing phase, where documents are initially divided into individual sentences. These sentences are then grouped based on their length to prevent short sentences from being treated as individual segments. Subsequently, the sentences are regrouped until they collectively reach the maximum token limit.

Additionally, a light normalization step is applied to the text data to ensure consistency, particularly with regard to units of measurement. For instance, engine parts' temperatures are standardized to include symbols like "°C".

Lastly, to validate answers, a function is developed to return a boolean value indicating whether the text qualifies as a correct answer. For an answer to be deemed valid, it must not be empty.

5.3 Prompt Engeenering and Extraction

Following the pre-processing phase, which involves data cleaning and segmentation of text into manageable chunks, the data is prepared for processing using Claude Instant.

For chunk selection, each chunk typically contains around 4000 tokens. To aid the LLM in identifying the requested keywords and achieve a more structured output, I provided a set of 'templates'. These templates functioned as dictionaries, with keys representing the desired information to be extracted and empty spaces (" ") as values, providing guidance to the transformers on what to search for and the references to follow.

The complete prompt was structured in the following way:

"Consider yourself an document assistant. Given the chunk containing a chapter on mechanical engines, your fundamental task is to populate a consistent key-value dictionary that fills in the empty values of the model using the present information from the chunk if you find it, otherwise leave a 'NA' value."

+

"One of the 10 TIPS prompt, based on the five tipologies of PDF file"

+

"One of the The dictionary template to populate based on the file being analyzed"

+

"I DO NOT want to see any Python code, explanations, or any strings outside of the dictionary. The output should consist ONLY of the dictionary I've requested, with the corresponding keys and values. Ensure that there are no duplicate keys in the output. Keys must be only what you see in 'field checklists' and the Values of the keys that you find in the chunk. Let's work this out in a step-by-step way to be sure we have the right answer."

Given the variability in document structures specified by the client, a universal prompt couldn't be applied. Instead, tailored prompts were created for each unique document type, resulting in a total of 10 distinct prompts. Each file type was associated with 2 purpose-built prompt types, offering specific guidance for optimal extraction.

5.4 Post-Processing

In this post-processing phase, various functions are employed to refine the information extracted from the LLM. One crucial function involves computing the average ratio for each chunk, the details of which are explained further below. The objective is to extract the relevant information from the tokens provided by the LLM. In cases where information is absent, the system is programmed to insert a 'Na', representing a Null value. To track the occurrence of Null values, an indicator is calculated by dividing the count of non-'Na' values by the total number of values. A higher ratio suggests a greater likelihood that the chunk contains the desired information.

5.5 Evaluation

The evaluation of extraction performance that Calude Instant implemented through the various files, starting from the client's guidelines, is defined as follows:

- **Non-extracted info**: only "nan" are present in the dictionary to be populated;
- **Incomplete info**: at least one correct value other than nan is present in the dictionary to be populated;
- **Complete info**: the dictionary is populated interemantly correctly.

Below I report the evaluation on three of the five types of files, as rows the file types and as columns the the scale of extraction completeness:

Tabella 5.1: Claude Istant Extraction Info - Evaluation

file type	**Non-extracted info**	**Incomplete Info**	**Complete Info**
CERT	8.237%	11.311%	80.452%
ESS	9.141%	16.322%	74.537%
MANUAL	15.814%	22.916%	61,270%

The results are very consistent on the different file types, we see how the best extraction performance is obtained on ESS type files and peaks on CERTs, while the high complexity and considerable thickness of the Manuals led to lower if overall decent performance. A dutiful contribution of the results was certainly made by the preprocessing phase in which OCR techniques with textract and tesseract made it possible to hook the documents to CLAUDE.
Finally, I report the table of costs incurred to implement the project [6]:

Tabella 5.2: Cost

	Claude Instant
Open Source	X
Cost per Hours	$39,60 (1 month) or $22,00 (6 month)
Cost per Tokens Input	$0,0008/1K tokens
Cost per Tokens Output	$0,0024/1K tokens
Max token per call	9k tokens
Number of call	$\sim 120k$
Total cost	$\sim$ $576

6. Conclusion and Future Developments

This chapter will draws conclusion. Sec. 6.1 will discuss the conclusions reached in this thesis. In Sec. 6.2, I will outline a step by step flowchart of the project goals achieved and settled by the comany. Fanally possible future developments to introduce, will be discussed in Sec. 6.3.

6.1 Conclusion

During this research, I delved into the field of synthetic data generation, data extraction and prompt engineering, highlighting the new role of Large Language Models (LLM) in the enterprise landscape. The thesis began with a state of the art, describing in detail the tools I used and the logic behind my choices. This work is anchored in two consultancy projects for companies: the first focused on the replication of data quality rules, while the second, focuses on the extraction of specific data present in the engine manuals.

I divided data processing into three phases: pre-processing, prompt engineering and data extraction. Initially, I cleaned and normalized the data, then manipulated it using various LLMs including GPT 3.5 - turbo, Claude V2.1 and Claude Instant. Finally I refined the model results through two metrics.

About Thesis Project reguarding synthetic structured data generation, I integrate an LLM in a data quality tool in order to replicate a JSON structure that identifies the configuration of a data quality rule. This goal was fully achieved, highlighting the ability of LLMs to generate consistent and contextualized output from natural language input. I validated the outputs with cosine similarity through different kor schema and template. We can see:

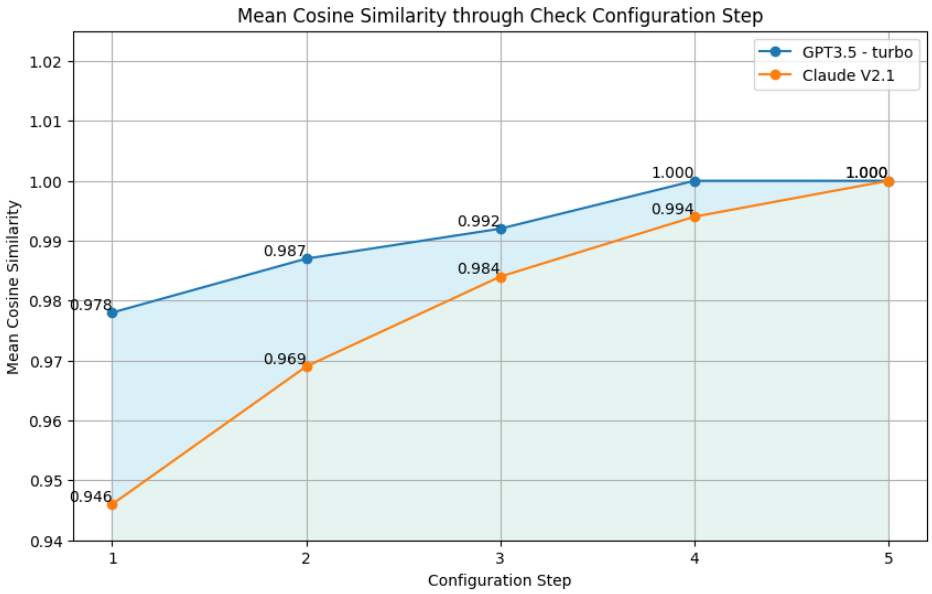


Figura 6.1: Performance Comparison - Data Quality Rule: Check Component

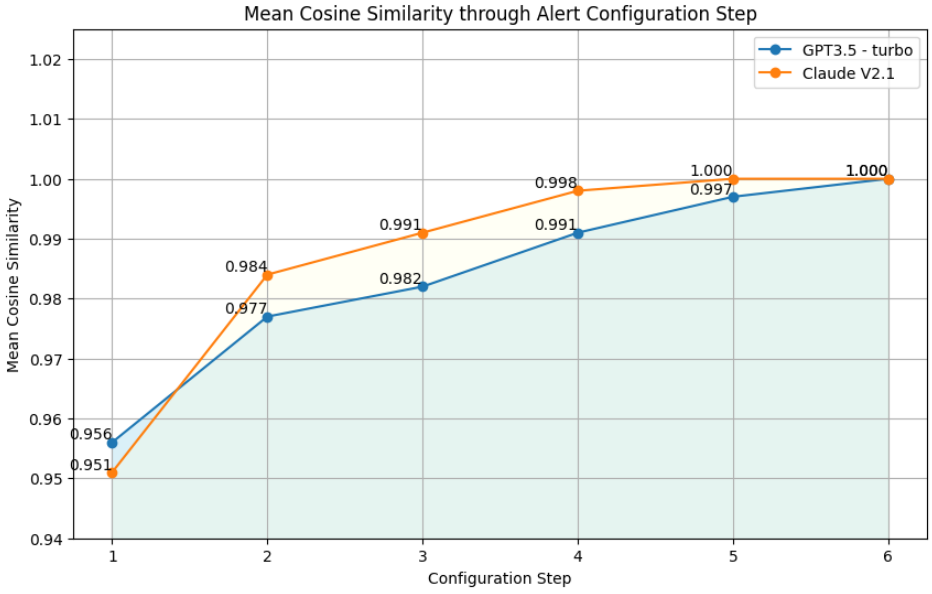


Figura 6.2: Performance Comparison - Data Quality Rule: Alert Component

While in the engine documents project, I created an indicator that evaluates the relative quantity of information extracted from files. The average recall is **72.086%**, in the following table we can see how Claude Instant performs:

Tabella 6.1: Claude Istant Extraction Info - Evaluation

file type	**Non-extracted info**	**Incomplete Info**	**Complete Info**
CERT	8.237%	11.311%	80.452%
ESS	9.141%	16.322%	74.537%
MANUAL	15.814%	22.916%	61.270%

The conclusions drawn from this thesis are not based exclusively on the performance of the model but I also took into account the computational costs, which deserve significant considerations for companies:

Tabella 6.2: Cost Thesis Project

	GPT3.5 - Turbo	**Claude V2.1**
Total cost	$\sim$ \$389	$\sim$ \$338

In the thesis project the GPT3.5 - turbo and Claude V2.1 models were explored, they can be found in the two main Web Services platforms: Azure and AWS. These two LLMs were opted for in order to give Reply an idea of which platform would be best used to put the final version of the chatbot and the new AlerTable ecosystem into production. The results are excellent for both models, as they manage to replicate the Json file for structuring the data quality rule, and furthermore the costs are decidedly low.

Tabella 6.3: Cost Intership Project

	Claude Instant
Total cost	$\sim$ \$576

In the internship project, however, it is clear how Claude Instant is a high-performance model for a data extraction task and how the parent company Anthropic is becoming a direct competitor of OpenAI, not only for performance but also for excellent costs.

This thesis highlights the extensive work surrounding structured data generation, OCR integration, langchain technologies, data acquisition amd data cleaning , **emphasizing that the process is not simply about creating a suggestion and leaving the rest to model**.

In conclusion, I consider it necessary to focus on two key topics of my thesis: data generation and Prompt Engineering. First of All, I wanted to convey how fundamental the role of prompt engineering is. In fact, many speculate that this could evolve into a distinct job role, overshadowing programming and traditional statistical skills, but based on my experiences throughout my studies, I believe that prompt engineering will instead be an essential skill. Secondly, with my thesis I wanted to highlight the contribution that generative artificial intelligence can make to the industrial landscape but at the same time raise everyone's awareness regarding synthetic data generation.

6.2 Flowchart

The following flowchart shows all the small successes achieved step by step during my Thesis and Intership experience in the reality of consultancy.

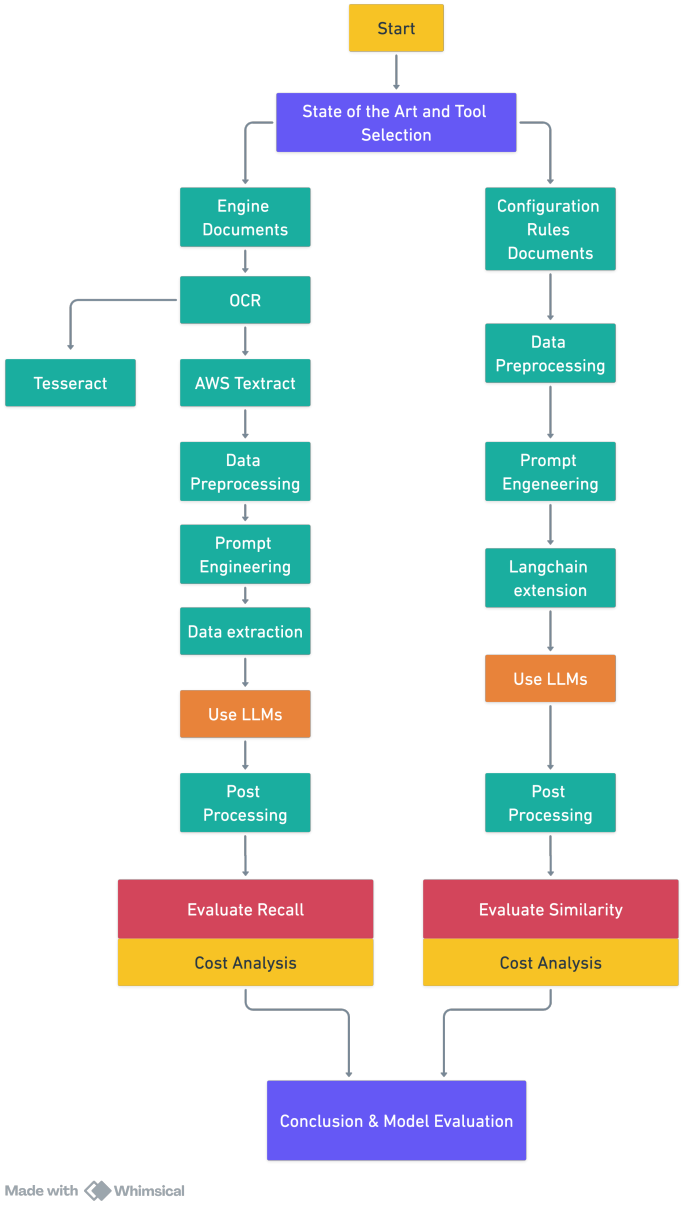


Figura 6.3: Goals Achieved - Flowchart

6.3 Future Development

These are some of the possible future developments:

- **Industrialization of the Model**:in the thesis, I extensively explored the generation and extraction of crucial information from data quality and engine manuals. The models and pipeline developed in this thesis were specifically tailored to these domains. However, the methodology could be applied to various other domains, opening up avenues for numerous new projects. For example, the kor langchain extension can be applied in the intership project.

- **English Documents**: using English documents for information extraction could potentially enhance performance. Language models tend to perform better with English text compared to other languages, such as Italian (the focus of my thesis), due to several factors: the abundance of data available in English, extensive research and development efforts, and the linguistic variations present in the English language.

- **New Models**: New Large Language Models are constantly released.

- **Explainable AI**: in addition to the traditional way of doing explainable through the fine tuning paradigm with local or global explaination, the LLMs have introduced a new way of doing explainability through the prompting paradigm [19].

Appendix A: Python Libraries

- NumPy[1]: is a library that provides a large collection of mathematical functions to facilitate the handling of large and/or multidimensional arrays and matrices.

- Pandas[2]: is a library for data manipulation and analysis. Specifically, it offers data structures and operations for manipulating numerical tables and time series.

- Langchain[3]: is a framework for developing applications powered by language models. There are two foundamental packages: langchain-community (used for third party integrations) and langchain (used for chains, agents, and retrieval strategies that make up an application's cognitive architecture).

- Kor[4]: integrated with the LangChain framework, this is a half-baked prototype that "helps" you extract structured data from text using LLMs . Specify the schema of what should be extracted and provide some examples. Kor will generate a prompt, send it to the specified LLM and parse out the output.

- SpaCy [5]: integrates Large Language Models (LLMs) into spaCy, featuring a modular system for fast prototyping and prompting, and turning unstructured responses into robust outputs for various NLP tasks, no training data required. Perfect to create strong pipline and validation your data.

- Seaborn[6]: è una libreria di visualizzazione dati basata su matplotlib. Fornisce un'interfaccia di alto livello per disegnare grafici statistici.

[1] https://numpy.org/
[2] https://pandas.pydata.org/
[3] https://python.langchain.com/docs/get$_started/introduction$
[4] https://eyurtsev.github.io/kor/
[5] https://spacy.io/
[6] https://seaborn.pydata.org/

- Tqdm[7]: the name comes from the Arabic word *taqaddum* which means ”progress” and is at the same time short for *te quiero demasiado* which means ”I love you so much” in Spanish . This library instantly shows the progress status of loops via a smart progress indicator.

- Warnings[8]:is a library for handling warnings in Python. Warning messages are typically issued in situations where exceptions are not handled by not allowing the program to be closed.

- Dill[9]: Library that allows you to save work sessions.

[7]https://pypi.org/project/tqdm/

[8]https://docs.python.org/3/library/warnings.html

[9]https://pypi.org/project/dill/

Appendix B: Check Synthetic Generation

```
llm = ChatOpenAI(
    model_name="gpt-3.5-turbo",
    temperature=0,
    max_tokens=4090,
    top_p = 1,
    openai_api_key=os.getenv('OPENAI_API_KEY') #openai_api_key
)

check_schema = Object(
    id="checks",
    description="Json Strutturato",

    # Notice I put multiple fields to pull out different
       attributes
    attributes=[
        Text(
            id="id",
            description="id della query, costruito con le parole
               chiave presenti nella query."
        ),
        Text(
            id="type",
            description="standard"
        ),
        Number(
            id="datasource_id",
```

```
    description="id■del■datasource,■{'covid19_italia
        ':198,■'london_merged':204,■'View':139,■'
        devices_daily':120,■'report_km_auto':184}"
),
Text(
    id="filter_function",
    description="funzione■applicata■alla■colonna■di■
        interesse■per■selezionare■le■righe■d'interesse,■la
        ■lista■delle■possibili■funzioni■  :■■['NO_FILTER',
        ■'CUSTOM_WHERE',■'COL_VALUES_EQUAL',■'
        COL_VALUES_GREATER',■'COL_VALUES_GREATER_OR_EQUAL
        ',■'COL_VALUES_LESS',■'COL_VALUES_LESS_OR_EQUAL',■
        'COL_VALUES_IN_SET',■'COL_VALUES_IN_EXT_SRC',■'
        COL_VALUES_BETWEEN',■'COL_VALUES_BETWEEN_LIST',■'
        COL_VALUES_LENGTH_EQUAL',■'
        COL_VALUES_LENGTH_BETWEEN',■'COL_VALUES_TYPE_EQUAL
        ',■'COL_VALUES_REGEX_MATCH',■'
        COL_VALUES_REGEX_MATCH_LIST',■'
        COL_VALUES_TIMESTAMP_FORMAT_MATCH',■'
        COL_PAIR_VALUES_A_EQUAL_B',■'
        COL_PAIR_VALUES_A_GREATER_B',■'
        COL_PAIR_VALUES_A_GREATER_OR_EQUAL_B',■'
        COL_PAIR_VALUES_A_LESS_B',■'
        COL_PAIR_VALUES_A_LESS_OR_EQUAL_B']■"
),
Text(
    id="filter_configuration",
    description="dizionario■composto■come■{'chiave':■'
        valore■per■filtrare'},■di■seguito■tutte■le■
        casistiche■del■dizionario■date■le■filter■function:
```

■COL_VALUES_TYPE_EQUAL:■{'type':'string'},■
COL_VALUES_IN_SET:■:■{'input_set':■array},■
COL_VALUES_IN_EXT_SRC:■{'external_source':■'string
',■'external_source_col':■'string'},■
COL_VALUES_TIMESTAMP_FORMAT_MATCH:■{'
timestamp_format':■'string'},■CUSTOM_WHERE:■{'
where_expr':■'sql'},■COL_VALUES_EQUAL:■{'value':■'
string'},■COL_VALUES_GREATER:■{'value':■number},■
COL_VALUES_GREATER_OR_EQUAL:■{'value':■number},■
COL_VALUES_LESS:■{'value':■number},■
COL_VALUES_LESS_OR_EQUAL:■{'value':■number},■
COL_VALUES_LENGTH_EQUAL:■{'len':■number},■
COL_VALUES_LENGTH_BETWEEN:■{'lower_bound':■number,
■'upper_bound':number},■COL_VALUES_REGEX_MATCH:■{'
regex':'string'},■COL_PAIR_VALUES_A_EQUAL_B:■{'
col_b':■'string'},■COL_PAIR_VALUES_A_GREATER_B:■{'
col_b':■'string'},■
COL_PAIR_VALUES_A_GREATER_OR_EQUAL_B:■{'col_b':■'
string'},■COL_PAIR_VALUES_A_LESS_B:■{'col_b':■'
string'},■COL_PAIR_VALUES_A_LESS_OR_EQUAL_B:■{'
col_b':■'string'},■NO_FILTER:■{},■AGG_COUNT_IF:■{'
expr':'string'},■AGG_COUNT_IF_RATIO:■{'expr':'
string'},■AGG_COUNT_NULL_VALUES:■{},■AGG_COUNT:■
{},■AGG_SUM:■{},■AGG_MAX:■{},■AGG_MIN:■{},■
AGG_MEAN:■{},■AGG_MEDIAN:■{},■AGG_QUANTILE:■{},■
AGG_STDDEV:■{},■AGG_COUNT_DUPLICATES:■{},■
AGG_COUNT_UNIQUE_VALUES:■{},■COL_VALUES_BETWEEN:■
{'lower_bound':■number,■'upper_bound':■number},■
COL_VALUES_REGEX_MATCH_LIST:■{'regex_list':■array}
"

```
        ),
        Text(
            id="aggregation_function",
            description="funzione di aggregazione applicata alla
                colonno di interesse , la lista delle possibili
                funzioni    : ['AGG_COUNT' , 'AGG_SUM' , 'AGG_MAX' , '
                AGG_MIN' , 'AGG_MEAN' , 'AGG_MEDIAN' , 'AGG_QUANTILE
                ' , 'AGG_STDDEV' , 'AGG_COUNT_DUPLICATES' , '
                AGG_COUNT_UNIQUE_VALUES' , 'AGG_COUNT_NULL_VALUES' ,
                'AGG_COUNT_IF' , 'AGG_COUNT_IF_RATIO']"
        ),
        Text(
            id="field_id",
            description="il nome della colonna a cui applicare le
                funzioni precedenti"
        ),
        Text(
            id="group_by",
            description="lista delle colonne in formato string
                per cui raggruppare"
        ),
        Text(
            id="description",
            description="query di input in linguaggio naturale"
        )
    ],
    many = True,
    examples=[
        (
            "Conteggio applicato al campo id del datasource
```

```
        report_km_auto.",
    [
        {
            "id": "Count_car_km_report_records",
            "type": "standard",
            "datasource_id": 184,
            "filter_function": "NO_FILTER",
            "filter_configuration": {},
            "aggregation_function": "AGG_COUNT",
            "field_id": "id",
            "group_by": [],
            "description": "Conteggio applicato al campo
                id del datasource report_km_auto"
        },
    ],
),
(
    "Conteggio dei record che hanno un valore di cnt >
        100 del datasource london_merged.",
    [
        {
            "id": "ck_count_cnt_greater_than_100",
            "type": "standard",
            "datasource_id": 204,
            "filter_function": "COL_VALUES_GREATER",
            "filter_configuration": {
                "value": 100.0
            },
            "aggregation_function": "AGG_COUNT",
            "field_id": "cnt",
```

```
                    "group_by": [],
                    "description": "Conteggio dei record che
                        hanno un valore di cnt > 100 del
                        datasource london_merged"
                },
            ],
        )
    ]
)


text="Conteggio dei record che hanno un valore di SNo < 40 del
    datasource covid19_italia , Conteggio dei record che hanno un
    valore di Eng_Percent_Torque > 25 del datasource devices_daily
    "
chain = create_extraction_chain(llm, check_schema)
output = chain.predict(text=text)['data']
printOutput(output)
```

Appendix C: Alert Synthetic Generation

```
llm = BedrockChat(
    model_id="anthropic.claude-v2.1",
    model_kwargs={"temperature": 0.1,
                  "max_tokens": 10000,
                  "top_p": 1}
)

alert_schema = Object(
    id="alert",
    description="Json Strutturato, se l'alert ha successo c' un
       problema di data quality",

    # Notice I put multiple fields to pull out different
       attributes
    attributes=[
        Text(
            id="id",
            description="il nome visualizzato dell alert, 
               costruito con le parole chiave presenti nell'input
               o scritto dall'utente."
        ),
        Text(
            id="checks_list",
            description="lista dei check dati dall'utente"
        ),
        Text(
```

```
            id="condition",
            description="condizione dell'alert definita dall'
                utente relativa al check selezionato. Dove le
                possibili function sono: AT_GET, AT_MAX, AT_MIN,
                AT_MEAN, AT_STD, AT_LR, AT_SUM. DOve i possibili
                field sono: success_count, fail_count, skip_count,
                null_count, success_ratio, fail_ratio, null_ratio
                , success_ratio_all, fail_ratio_all,
                skip_ratio_all, null_ratio_all, mean. Gli
                operatori matematici sono: >, <, >=, <=, ==, !=.
                index_baseline    il filtro temporale relativo al
                tempo del data source"
        ),
        Text(
            id="message",
            description="frase di risposta in caso di successo
                dell'alert condition"
        ),
        Number(
            id="severity",
            description="importanza del controllo"
        ),
        Text(
            id="value",
            description="null"
        )
    ],
    many=True,
    examples=[
        (
```

```
    "Dato il check_eng_Percent_Torque verifica che il
       numero di device che hanno una coppia motore (%),
       registrata ieri , sia maggiore rispetto a 0.",
    [
        {
            "id": "alt_check_eng_Percent_Torque",
            "checks_list": ["check_eng_Percent_Torque"],
            "condition": "AT_GET(\n\tcheck_id=\'
               check_eng_Percent_Torque\',\n\tfield=\'
               success_count\',\n\tindex_baseline=\';\',\
               n\tindex_run=\'0;0\'\n) > 0",
            "message": "Il numero di device che hanno una
               coppia motore (%), registrata ieri ,
               maggiore rispetto alla soglia impostata
               (%)",
            "severity": 1,
            "value": 'null'
        },
    ]
),
(
    "Il conteggio giornaliero    inferiore per pi  del
       80 percento della media nell'ultima settimana ,
       utilizza il check_id Count_car_km_report_records",
    [
        {
            "id": "counting_of_car_km_report",
            "checks_list": ["Count_car_km_report_records"
               ],
            "condition": "AT_GET(\n\tcheck_id=\"
```

```
                        Count_car_km_report_records\",\n\tfield=\"
                        success_count\",\n\tindex_baseline=\";\",\
                        n\tindex_run=\"0;0\"\n) < AT_MEAN(\n\
                        tcheck_id=\"Count_car_km_report_records
                        \",\n\tfield=\"success_count\",\n\
                        tindex_baseline=\";\",\n\tindex_run
                        =\"7;1\"\n)*0.8",
                    "message": "Il conteggio giornaliero
                        inferiore per pi  del 80 percento della
                        media nell'ultima settimana",
                    "severity": 1,
                    "value": 'null'
                },
            ]
        ),

    ]
)

text="Crea un alert che verifica se esistono delle polizze
   presenti nello storico che non sono presenti nella tabella
   online dato il check ck_unipol"

chain = create_extraction_chain(llm, alert_schema)
output = chain.predict(text=text)['data']

printOutput(output)
```

Bibliografia

[1] Daniel J. Abadi. «CAP Twelve Years Later: How the ”Rules” Have Changed». In: *Computer* (2012).

[2] M. M. Alsheikh e et al. «Data Masking Using Generative Adversarial Networks for Anomaly Detection in Smart Grids». In: (2018).

[3] Amazon Web Services. *Amazon Textract Documentation*. URL: `https://docs.aws.amazon.com/managedservices/latest/userguide/textract.html`.

[4] Amazon Web Services. *AWS Bedrock*. URL: `https://aws.amazon.com/it/bedrock/`.

[5] *Anthropic*. URL: `https://www.anthropic.com/research`.

[6] *AWS Price*. URL: `https://aws.amazon.com/it/bedrock/pricing/`.

[7] P. Azad et al. «The Role of Structured and Unstructured Data Managing Mechanisms in the Internet of Things». In: *Cluster Computing* ().

[8] A. Baevski et al. «Language Understanding with Transformers». In: ().

[9] Yassine Benajiba, Yassine Maâmar e Hamid Bessaid. «Named Entity Recognition: A Literature Review». In: (2021). URL: `https://www.researchgate.net/publication/348863897_Named_Entity_Recognition_A_Literature_Review`.

[10] Eric A. Brewer. «Brewer’s Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services». In: *ACM SIGACT News* (2000).

[11] T. B. Brown e et al. «Language Models are Few-Shot Learners». In: *Advances in Neural Information Processing Systems 33 (NIPS 2020)*. 2020.

[12] T. B. Brown e et al. *Language Models are Unsupervised Multitask Learners*. OpenAI. 2019.

[13] S. Büttcher, C. L. A. Clarke e G. V. Cormack. *Information Retrieval: Implementing and Evaluating Search Engines*. 2016.

[14] J. Cook. «How to write effective prompts for ChatGPT: 7 essential steps for best results». In: *Forbes* (set. 2023).

[15] Shubham Dandwate et al. ≪Synthetic Data Generation Using Generative Adversarial Networks for Semantic Segmentation of Satellite Images≫. In: (2020). URL: https://ieeexplore.ieee.org/document/9152280.

[16] T. Dettmers et al. *LLM matrix multiplication for transformers at scale*. https://arxiv.org/pdf/2208.07339.pdf. 2022.

[17] J. Devlin e et al. ≪BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding≫. In: *arXiv preprint arXiv:1810.04805* (2018).

[18] Ian Goodfellow et al. ≪Generative Adversarial Nets≫. In: *Advances in neural information processing systems*. 2014.

[19] Hanjie Chen Haiyan Zhao et al. ≪Explainability for Large Language Models: A Survey≫. In: (2023). URL: https://arxiv.org/abs/2309.01029.

[20] Ali Hassani et al. ≪Synthetic Data Generation: A Must-Have Skill for New Generative Models≫. In: (2019). URL: https://arxiv.org/abs/1910.03876.

[21] K. Hebenstreit et al. ≪An automatically discovered chain-of-thought prompt generalizes to novel models and datasets≫. In: *arXiv preprint arXiv:2305.02897* (2023). Available at: https://arxiv.org/abs/2305.02897.

[22] S. Hochreiter e J. Schmidhuber. ≪Long Short-Term Memory≫. In: (1997).

[23] A. Hodges. *Alan Turing: The Enigma: The Book That Inspired the Film "The Imitation Game"*. Available at: https://www.degruyter.com/document/doi/10.1515/9781400865123/html. Princeton University Press, 2014.

[24] *Kor*. URL: https://eyurtsev.github.io/kor/tutorial.html.

[25] *Langchain*. URL: https://python.langchain.com/docs/get_started.

[26] Yann LeCun, Yoshua Bengio e Geoffrey Hinton. ≪Deep learning≫. In: *Nature* ().

[27] M. Lewis e et al. ≪BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension≫. In: *arXiv preprint arXiv:1910.13461* (2019).

[28] Y. Liu e et al. ≪RoBERTa: A Robustly Optimized BERT Pretraining Approach≫. In: *arXiv preprint arXiv:1907.11692* (2019).

[29] Professore Piergiorgio Lovaglio. *Materiale del corso di Data Mining e Statistica Computazionale*. 2020.

[30] Yingzhou Lu et al. ≪Machine Learning for Synthetic Data Generation: A Review≫. In: (2023).

[31] Christopher D Manning, Prabhakar Raghavan e Hinrich Schütze. *Introduction to Information Retrieval*. 2008.

[32] Mitchell P Marcus, Beatrice Santorini e Mary Ann Marcinkiewicz. *Part-of-speech tagging guidelines for the Penn Treebank Project (3rd Revision)*. Rapp. tecn. University of Pennsylvania, 1993.

[33] T. Mikolov e et al. ≪Distributed Representations of Words and Phrases and their Compositionality≫. In: (2013).

[34] S. Muruganandam et al. ≪A deep learning based feed forward artificial neural network≫. In: *ScienceDirect* (feb. 2023). Available at: https://www.sciencedirect.com/science/article/pii/S2665917422002471.

[35] Gourab Nath e Amlan Chakrabarti. ≪A Review of Machine Learning Techniques for Predictive Analytics≫. In: (2017). URL: https://arxiv.org/abs/1708.05070.

[36] *OpenAI Price*. URL: https://openai.com/pricing.

[37] *Output Quality*. URL: https://medium.com/morningside-ai/cosine-similarity-our-secret-weapon-for-output-quality-control-8a68cfd0fa5b.

[38] G. Pinei. ≪A comparison of SageMaker and Databricks for machine learning≫. In: (lug. 2023).

[39] *PromptEngineering*. URL: https://learnprompting.org/docs/intro.

[40] *Python*. URL: https://www.python.org/.

[41] A. Radford e et al. *Improving Language Understanding by Generative Pre-Training*. OpenAI. 2018.

[42] A. Radford e et al. «Language Models are Few-Shot Learners». In: *Advances in Neural Information Processing Systems 33 (NIPS 2020)*. 2020.

[43] C. Raffel e et al. «Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer». In: *Journal of Machine Learning Research* 22.140 (2021), pp. 1–67.

[44] N. Rambabu et al. «Generating Synthetic Data for Neural Network Training using Generative Adversarial Networks». In: *International Journal of Engineering Research and Technology* (). URL: https://www.ijert.org/research/generating-synthetic-data-for-neural-network-training-using-generative-adversarial-networks-IJERTV10IS070098.pdf.

[45] Adwait Ratnaparkhi. «A Maximum Entropy Approach to Natural Language Processing». In: *Journal of Artificial Intelligence Research* ().

[46] Olesya Razuvayevskaya et al. «Comparison between Parameter-Efficient Techniques and Full Fine-Tuning: A Case Study on Multilingual News Article Classification». In: (2023).

[47] *Reply Company Profile*. URL: https://www.reply.com/it/company-profile.

[48] Frank Rosenblatt. *The perceptron: a probabilistic model for information storage and organization in the brain.* 1958.

[49] N. Shazeer e et al. *BERT: Bidirectional Encoder Representations from Transformers*. arXiv preprint arXiv:1810.04805. 2018.

[50] K. Sims. «Ultimate OpenAI Playground Settings Guide - WordBot». In: (feb. 2023).

[51] K. Sparck Jones. «Natural Language Processing: A Historical Review». In: *Springer* (2021). Available at: https://link.springer.com/chapter/10.1007/978-981-15-9712-1_31.

[52] T. Sun et al. «Paradigm Shift in Natural Language Processing». In: *Medium* (2023). Available at: https://medium.com/search?q=An+Attempt+to+Chart+the+History+of+NLP+in+5+Papers.

[53] A. Vaswani e et al. «Attention is All You Need». In: *Advances in Neural Information Processing Systems 30 (NIPS 2017)*. 2017.

[54] S. Vibhor, M. Goyal e M. Drishti. *Eliza: First chatbot*. 2017. URL: https://www.neliti.com/publications/263312/an-intelligent-behaviour-shown-by-chatbot-system.