UNIVERSITÀ DI PISA

DIPARTIMENTO DI INFORMATICA

Corso di Laurea Magistrale in Data Science and Business Informatics

TESI DI LAUREA

AN ADVANCED ANALYTICS FRAMEWORK FOR CUSTOMER CARE ANALYSIS: EXPLORING BUSINESS ENHANCEMENT THROUGH LLMS

RELATORE

Prof. Claudio GALLICCHIO

CANDIDATA

Chiara DE NIGRIS

ANNO ACCADEMICO 2022-2023

The present work provides an overview of the use of Generative Large Language Models as tools for extracting features from customer service chats in a corporate setting, integrating models from the GPT family into an advanced analytics architecture. This study gains significance through the development of an Italian-language prompt for feature extraction, utilizing state-of-the-art prompt engineering techniques. The proposed experiments not only showcase the achieved advancements but also shed light on the primary challenges tied to integrating a model of this nature into an analytics system. Particularly intriguing in the context of this research is the incorporation of Language Models into a business environment, probing the delicate balance between expenses, efficiency, and consistency. The proposed experiments underscore the significant leap forward in modernizing and streamlining analytical procedures that come with integrating generative models into a business process, returning suitable outcomes even without the need for model specialization or the reliance on contextually labeled examples. Even though satisfactory results have already been achieved, the ongoing experimental configuration of the proposed system allows for further improvement in the future by exploiting cutting-edge methodologies. It is evident from the experiments that harnessing the great power of these tools must be coupled with thoughtful assessments from a human perspective. Validation of outputs generated by a model of this nature by experts becomes imperative, highlighting the essential need to blend the capabilities of these advanced tools with the nuanced insights provided by human expertise.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

The present work provides an overview of an advanced analytics framework developed for the analysis of customer service conversations in a business context. The innovative aspect of the project is the integration of Natural Language Processing systems, such as the generative Large Language Models (LLMs) of the GPT family, as the main tool within the project's architecture, entrusting Artificial Intelligence with the task of extracting predefined features from conversations in the Italian language.

The case study was conducted at the company Advant, involving collaboration with a client company operating in the legal public gaming sector in Italy. The presented project serves as an experimentation carried out in collaboration with the client, aiming to consolidate the achieved results into a more robust structure as quickly as possible. This consolidation is intended to integrate the analysis framework among the tools used by the company for its advanced analytics.

Assuming the experimental character of the use case, the primary contribution of the presented work was to test the best possible solutions for the feature extraction task. For the experiments it has been defined an architecture entirely developed on the Cloud, utilizing the services provided by Microsoft Azure in all its phases, designed coherently with an advanced analytics architecture. The most innovative aspect of the project lies in the utilization of Generative Large Language Models to extract interesting features from chats, which required the design of instructions in natural language for their interaction. One of the main contributions of the work in this regard was the crafting of a suitable prompt for the feature extraction task, employing different experiments that integrated cutting-edge techniques and methods to achieve the best results possible. The ultimate goal was to create a clear, concise, and well-structured prompt to perform the task of feature extraction from Italian language texts with performance comparable to human capabilities.

Particularly intriguing in the context of these experiments has been understanding how to integrate state-of-the-art model experiments into a business context. In fact, within a corporate setting, considerations must be made for more stringent cost management, integration of analyses into existing frameworks and the development of systems usable in real-time analysis. These factors have been taken into account in the development of the advanced analytics system, where implementation had to strike a balance between costs, performance, speed and reproducibility.

1.1 Advant

Advant[1] is a consulting and System Integration company founded in 2014 to provide professional services in the field of Information Technology and is part of the Agic group, one of the main Microsoft Italian sponsors. With main offices in Rome and Milan, the company has several competence centers throughout Italy and internationally, with disparate clients in both the private sector and Public Administration.

The company guides its clients through all stages of digital transformation, using the best market technologies both on-premise and in the Cloud, provided by major players such as Microsoft, Oracle, Databricks, SAS, and Tableau.

The main competence centers of Advant are:

[1]Advant official website: `https://advant.it/`.

- business Analytics;
- corporate Performance Management.
- data Integration;
- cata Warehousing;
- business Intelligence;
- big Data.

Among the services offered, the Data and Artificial Intelligence business unit aims to create Business and Artificial Intelligence systems by combining data analytics systems with the application of sophisticated Artificial Intelligence algorithms applied to tasks such as Machine Learning for recognition, forecasting, and Bot and RPA (Robotic Process Automation) systems.

Consistent with the digital migration of the majority of the companies nowadays, through the use of business intelligence and analytics applications, Advant aims to improve companies business performance by maximizing their Key Performance Indicators (KPIs) through the application of state of art data analytics techniques. Experimentations that are presented in this work have been conducted for Advant's client company leading a group of entities active in the market of authorized legal public gaming.

1.2 Chapters Overview

The present work is structured as follows: after the current introductory chapter, Chapter 2 delves into the fundamental concepts related to the field of Advanced Data Analytics. Advanced analytics encompasses sophisticated analysis techniques that integrate modern Machine Learning and Artificial Intelligence approaches into traditional data analysis processes, adapting to larger volumes and varieties of data from heterogeneous sources. In line with the treated use case, the chapter outlines the structure of an advanced analytics architecture, a system designed to support rapid processes involving vast amounts of data. To meet these demands, such analyses are typically conducted by leveraging the advantages of Cloud computing, as proposed in this use case which relies on Microsoft Azure services.

Chapter 3 provides an overview of the State of the Art regarding Large Language Models, which are crucial tools for the use case. The concept of language modeling is essential in enabling machines to process natural language input and output by predicting the probabilities associated with sequences of words. The Large Language Models that have gained prominence nowadays are tools that facilitate the execution of language-related tasks with results comparable to human cognitive abilities. For this use case are employed language generative models from OpenAI, whose architecture and functionality are detailed in the chapter. The interaction with these models is developed through the crafting of instructions in natural language, emphasizing prompt engineering and exploring possible techniques for shaping the instructions.

The subsequent Chapter 4 introduces the design of the proposed solution for the use case. Starting with a more precise definition of the project, the provided input data is described, consisting of a sample of customer service chat interactions in Italian from the company stored in a CSV file. After cleaning and manipulating the provided data to ensure its full usability and obtaining a comprehensive understanding of the data usage, statistical analyses of the conversations were conducted. Subsequently, the nine features to be extracted from the conversations as the desired output are defined. This process involved continuous collaboration with the client company. The chapter also drafts the architecture of the use case, developed using Microsoft Azure services.

Chapter 5 outlines the development of the designed solution. The initial phase involved the selection of the OpenAI Large Language Model to be used from those available with the Azure OpenAI service, leading to the choice of GPT-4-Turbo. A crucial aspect of the experimentation was the crafting of the prompt to guide the model in the feature extraction task. To write the instruction, various state-of-the-art techniques were tested, and several best practices were incorporated, defining also an evaluation scheme comparing the results to human annotations. The chapter describes the actual workflow for feature extraction, starting from input manipulation, interacting with the model through Open AI API and shaping the generated output to achieve the desired result.

Chapter 6 presents an evaluation of the obtained results, focusing on key insights derived from initial experiments. Essential to this assessment is the definition of potential future developments for the use case, intending to evolve from a mere experiment into a production analysis system integrated with the existing company analytics.

Chapter 7 contains the main conclusions drawn from the illustrated experiments and the contribution provided by the work in a business analytics context.

All the code written for the illustrated experiments can be consulted within the various sections of the Appendix A.

Chapter 2

BACKGROUND ON ADVANCED DATA ANALYTICS

This chapter elucidates the fundamental characteristics supporting modern data analytics techniques. Section 2.1 introduces data analysis techniques concerning large volumes of data characterized by variety and volume, commonly known as big data. It also highlights the concept of advanced analytics, encompassing sophisticated analysis techniques linked to state-of-the-art Artificial Intelligence and Machine Learning technologies, such as those employed in this use case. Within this section, Subsection 2.1.1 defines the essential features of the types of data encountered in a contemporary analysis process. Section 2.2 delineates the composition of an advanced analytics architecture, focusing on the data ingestion phase in Subsection 2.2.1 and on data storing in Subsection 2.2.2. The last Section 2.3 aims to provide a detailed explanation of the Cloud services offered by Microsoft used to develop the current project. Subsection 2.3.1 provides general background on the concept of Cloud computing, while Subsection 2.3.2 outlines the fundamental features of Microsoft Azure, with a specific focus on some services used in the project.

2.1 Big Data Analytics and Advanced Analytics

Data analytics is the process of data-driven knowledge and information extraction through the application of different techniques and technologies to draw significant conclusions. In essence, from raw data collected from heterogeneous sources we can extract common patterns, relationships, correlations or anomalies, which are information that can be used for a wide range of different purposes. Data analytics leverages mathematical and statistical modeling techniques combined with knowledge from computer science and information engineering.

In today's context, when we discuss data analysis we are especially referring to the examination of vast amounts of information derived from diverse sources and various formats, the so-called big data. Big Data Analytics pertains to the application of data analytics techniques to data marked by substantial variety, velocity, and volume [Elgendy and Elragal, 2014]. This shift has necessitated adjustments in analysis architectures and tools, as traditional techniques are becoming progressively difficult to apply. From an infrastructural perspective, there has been a need to adapt storage not only to accommodate vast amounts of data but also to expedite their retrieval. Analyses applied to these types of data typically aim to be real-time or very fast, requiring infrastructural adjustments at various levels. Crucial for overcoming computational complexity and speeding up analysis execution has been the widespread adoption of Cloud computing technologies (see Section 2.3), which enables overcoming computational limits by relying on extensive storage spaces, quick responses, and real-time collaboration possibilities for multiple users [Al-Jumaili et al., 2023]. The benefits provided by Cloud computing, including elasticity, a pay-per-use economic model and minimal upfront investment, have established it as a practical and attractive choice for the storage, management and analytics of big data [Berisha et al., 2022]. Today, major companies in the IT landscape, such as Microsoft, Amazon and Google, offer platforms for developing comprehensive big data analysis, integrating modern analysis algorithms proficiently and cost-effectively.

In synergy with the increasing production of big data, there has been the development of technological revolutions such as Machine Learning and Artificial Intelligence algorithms, which have been essential to integrate as tools for sophisticated analysis. The term Advanced Analytics encompasses a spectrum of refined techniques within the domain of data science, surpassing the limitations of traditional methods. In point of fact, in comparison to standard business intelligence analysis, data analytics offers not only descriptive analysis but encompasses various types of analyses. Indeed, advanced analysis can be predictive, foreseeing the future development of a situation, descriptive, aimed at uncovering patterns in historical data, and prescriptive, involving the resolution of specific problems. Advanced analytics techniques leverage a heterogeneous set of tools and methodologies, including Artificial Intelligence, Machine Learning and Deep Learning. Additionally, Natural Language Processing, Data Mining techniques and statistical analyses contribute to the robust framework that sustains the exploration and extraction of valuable insights from the thriving landscape of big data [Bose, 2009]. Examples of such techniques are clusterization and cluster analysis, to group data with similar characteristics, time series analysis, to analyze data over time, data visualization, to present in a more direct way the information found, text classification, to define the argument of a text. These techniques play pivotal roles in extracting meaningful insights from extensive datasets. Among them, clusterization and cluster analysis, for instance, involve grouping data with similar characteristics, facilitating a deeper understanding of patterns and relationships within the information, while time series analysis is instrumental in scrutinizing data trends and patterns over time. Data visualization, on the other hand, provides a more intuitive and direct representation of the discovered information. Through visualizations, complex datasets can be translated into easily comprehensible graphics, aiding stakeholders in making informed decisions. Other techniques are related to the realm of Natural Language Processing, such as text classification or sentiment analysis and are elaborated more comprehensively in Section 3.1.

Applying advanced data analytics techniques in a business context enables the realization of numerous benefits, ranging from the impact of data on decision-making processes to the development of customized and customer-oriented systems. These types of analyses allow companies to identify specific patterns in the behavior of their customers, highlight potential issues, and enhance cost-effectiveness and efficiency [Berisha et al., 2022]. Data analytics techniques apply to several business contexts, from marketing or sales to sectors such as healthcare, banking, energy or education, expanding the horizons of numerous fields.

2.1.1 Advanced Analytics Data

Advanced analytical processes involve complex data that is volatile, voluminous and diverse. The intricacies of managing heterogeneous data sources, varying formats and dimensions highlight the sophisticated nature of advanced analytics. It underscores the significance of employing focused manipulations to uncover the wealth of information hidden within the data, ultimately facilitating the achievement of diverse objectives.

In light of these premises, it becomes imperative to distinguish the different types of data based on their format. This categorization is crucial because different manipulations are required based on the type of data in order to extract the most embedded latent knowledge possible. Data can be distinguished in:

1. *Structured*: structured data refer to information organized in a precise pre-defined data format, usually in rows and columns. They come with an associated data model and can be easily inserted into traditional relational database storage

[Mishra and Misra, 2017]. Structured data are the easiest to store, manipulate and use for business analysis.

2. *Unstructured*: in contrast to structured data, the majority of big data that nowadays are used for analysis can not be connected with a pre-defined format or be stored in a relational DB [Mishra and Misra, 2017]. Unstructured data are assumed to be faster-growing concerning structured ones and nowadays are mostly analyzed by Artificial Intelligence and Machine Learning algorithms, which make the manipulation operations easiest. Examples of such types of data are texts, social media content, images or call transcriptions.

3. *Semi-structured*: beyond structure and unstructure data there are semi-structured ones, which have some consistent features, but do not adhere to a structure as rigid as what is typically expected in a relational database. Semi-structured data are fluids as unstructured ones but present some tags or metadata that guarantee a certain organization in their structure. Examples of these types of data are email messages, JSON or XML files.

Data analytics pipelines have to handle the navigation and storage of extensive datasets with dynamic characteristics, often integrating structure, semi-structured, and unstructured data in the same analytical framework. Each piece of data demands tailored manipulation strategies to reveal the insights it holds and derive the specific knowledge required to meet distinct analytical goals.

2.2 Advanced analytics architecture

Within the framework of a contemporary data analytics architecture, there exists a structured sequence of components designed to handle diverse aspects of the analytical process, which are shown in Figure 2.1. The initial stage entails data ingestion on which further details are provided in the subsequent Section 2.2.1. In order to

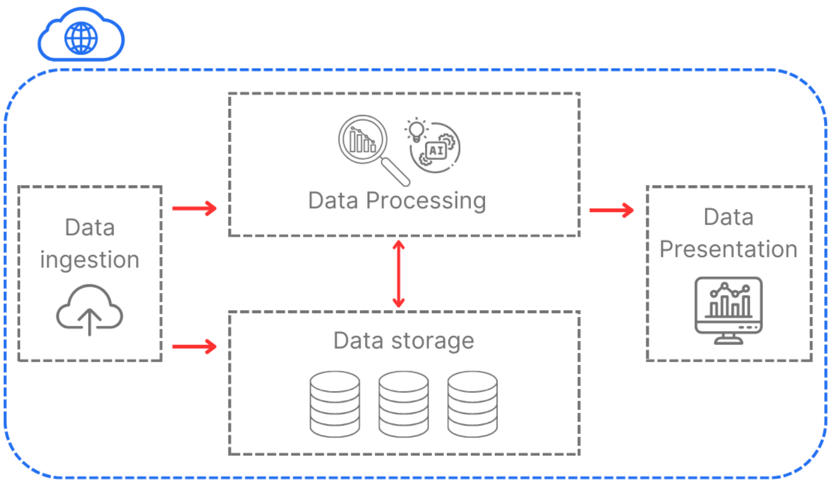


Figure 2.1: Example of a general Advanced Analytics architecture.

ensure a streamlined and organized process, data ingestion is typically managed by a dedicated orchestration tool that coordinates the seamless flow of data into the analytics system. Following data ingestion, the architecture incorporates one or more storage elements, which serve as the foundation for housing and managing the datasets that fuel the analytics pipeline. In this kind of process, storage elements involved are typically data warehouses or data lakes, which are better explained in Section 2.2.2. The subsequent phase is data processing, where the ingested data undergoes various transformations, analyses, and computations. This stage is fundamental in extracting meaningful insights from raw data, and it often involves the application of sophisticated Machine Learning, Artificial Intelligence algorithms and analytical techniques. The final component of the architecture is focused on presenting data in a way that is easy to understand. Visualization tools and reports are often used to effectively communicate insights to stakeholders, facilitating informed decision-making. These phases operate in constant interaction, forming a dynamic and iterative cycle within the data analytics architecture. This interconnectedness ensures that insights gained from one phase can inform and enhance subsequent stages, fostering a continuous improvement loop in the analytical process [Microsoft, b].

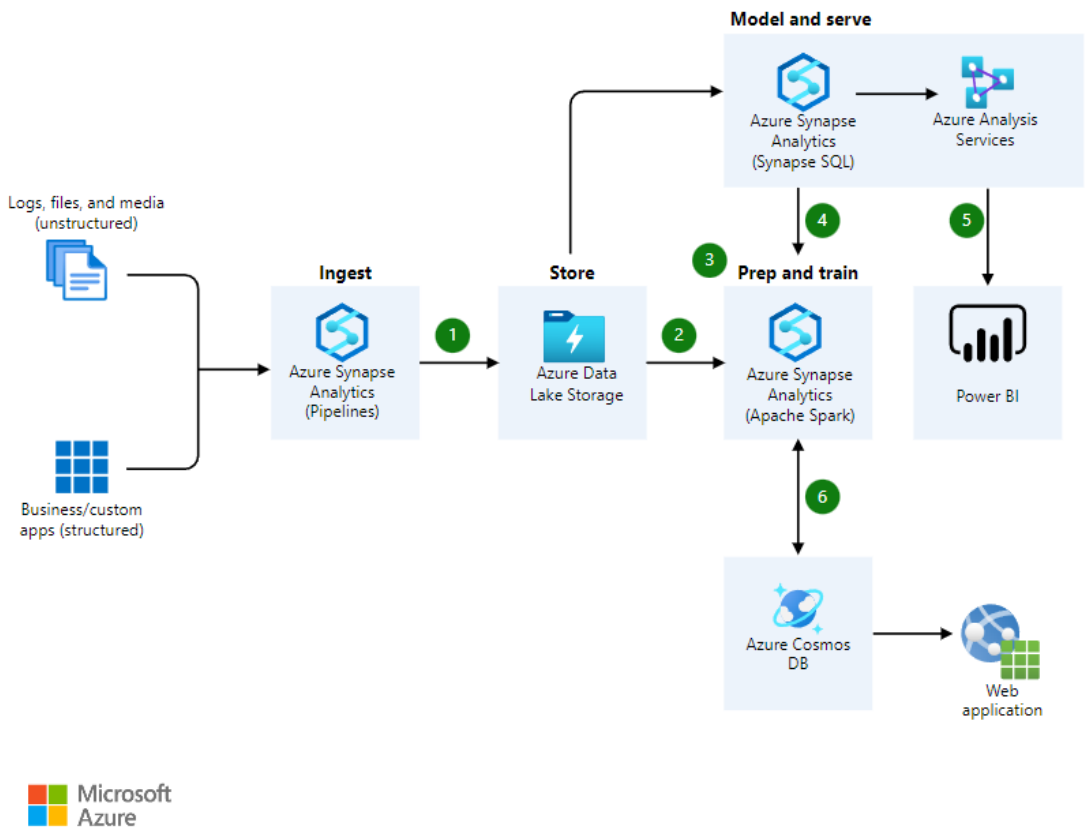

Model and serve
Azure Synapse Analytics (Synapse SQL)
Azure Analysis Services
Logs, files, and media (unstructured)
Ingest
Store
Prep and train
Azure Synapse Analytics (Pipelines)
Azure Data Lake Storage
Azure Synapse Analytics (Apache Spark)
Power BI
1
2
3
4
5
6
Business/custom apps (structured)
Azure Cosmos DB
Web application
Microsoft Azure

Figure 2.2: A Modern Advanced Analytics Architecture using Microsoft Azure Cloud services [Microsoft, a].

In contrast to traditional analytical systems that struggle with specific workloads, modern architectures harness scalable Cloud services with the capability to manage any volume of data, support multiple users on a single platform and shift from conventional descriptive analyses to more innovative predictive analyses. As mentioned in the previous Section 2.1, indeed the most renowned vendors provide all the tools to develop a Cloud-based data analytics architecture. For example, Figure 2.2 provides an overview of a typical contemporary analytics architecture that leverages Microsoft Azure Cloud services. The specific services and features related to Microsoft Azure are more comprehensively elucidated in the subsequent Section 2.3.

2.2.1 Data ingestion: ETL and ELT processes

Data ingestion involves the intricate process of importing vast and diverse data files from multiple sources and seamlessly integrating them into the analytic architecture. The conventional method of integration, known as the ETL (Extract, Transform, Load) process, acts as an intermediary step between the input and output of the analysis process. This process entails transferring data from one storage to another after applying specific manipulations to enhance its utility.

In recent years, an alternative approach known as ELT (Extract, Load, Transform) has replaced ETL both in terms of processes and software definitions. Indeed, in the ETL paradigm, data is transformed within the source system, typically a database. Consequently, extraction and transformation operations are performed on such a sysftem, which significantly impacts performance. The data is then loaded into a data warehouse or a data lake. Given the current affordability of Cloud services, there has been a shift towards ELT paradigm, which strategically shifts the most compute-intensive activities and transformations away from on-premise services that are often burdened with regular transaction-handling pressures. Instead, this approach leverages the capabilities of the Cloud to perform transformations, eliminating the need for data staging. This is particularly advantageous when dealing with diverse data types, including structured, semi-structured, unstructured, and raw data [Berisha et al., 2022]. In contemporary discussions, the concept of ELTL has gained prominence. This approach involves a nuanced process that includes the extraction and loading of data into an interim staging area. Subsequently, the data undergoes a series of transformations before being reloaded into the final destination. This architectural shift goes beyond the confines of traditional business intelligence and expands its applicability to advanced analytics.

Today, there are various tools and Cloud services for ELT and ETL; famous examples include Microsoft SQL Server Integration Services (SSIS), SAS Data Management, or, focusing on Cloud services, Azure Data Factory.

2.2.2 Data storage

There are several different structures in which data can be stored. Usually, in the context of Data analytics, we commonly refer to Data Warehouses (DWs) and Data Lakes (DLs). According to the definition of William Inmon, a “Data Warehouse is a subject-oriented, integrated, nonvolatile, and time-varying collection of data in support of management’s decisions” [Inmon, 2002]. Data Warehouses strategically fulfill a crucial role in data analytics storage, leveraging their capability to furnish a unified, well-structured, and optimized environment for data storage. Indeed, data modeling facilitates efficient analysis, informed decision-making and a more profound comprehension of an organization’s data landscape.

However, the increase in semi-structured and unstructured data has led to the need for storage repositories capable of accommodating raw and unprocessed data in its original format [Nambiar and Mundra, 2022]; these kinds of storage are called Data Lakes. Echoing Dixon’s definition in [Dixon, 2010], “while a data warehouse seems to be a bottle of water cleaned and ready for consumption, then a Data Lake is considered as a whole lake of data in a more natural state.” Data Lakes are highly flexible repositories that ingest and store large volumes of data of different sizes and from different sources, such as database tables, web pages or social media content. Modern data analytics architectures embrace the concept of data lakes because they do not require the pre-transformation of data before loading. This flexibility allows for the storage and analysis of various data formats without the need for rigid preprocessing, providing a versatile and efficient solution for handling the evolving landscape of data in analytics.

Even if DWs and DLS store and process data in a similar way, one of the main differences between them is that while Data Warehouses contain only filtered and processed data, Data Lakes allow the storage of raw data, processing it only when necessary. Furthermore, they have different use cases: a Data Warehouse is usually implemented to meet specific business needs, while the purpose of a Data Lake is not known from the outset [Microsoft, c].

2.3 Microsoft Cloud services for advanced analytics

As elucidated in the preceding sections, the orchestration of Cloud-based architecture is increasingly prevalent in the context of advanced data analytics. The advantages that companies can derive from transitioning their business to the Cloud are numerous, encompassing improvements in speed, cost-effectiveness, security and performance. Beyond the tangible benefits, the decision to conduct analyses in the Cloud often becomes a necessity, tied to the scale of the data at hand, the required analysis speed and the shared management of services among multiple users.

In contemporary times, major technology companies provide Cloud services, with Microsoft Azure being one of the most prominent providers. The services of Microsoft Azure are utilized for the implementation of this project and are elaborated upon in the subsequent subsections. Other noteworthy providers include Amazon Web Services (AWS), Google Cloud Platform (GCP) and IBM.

2.3.1 Background on Cloud Computing

Cloud computing is a global computing paradigm consisting of the on-demand delivery of services provided through the internet (the "Cloud") by a provider on a pay-as-you-go basis. According to the National Institute of Standards and Technology of USA [Cloud, 2011], Cloud computing presents five essential characteristics: *on-demand self-service*, *broad network access*, *resource pooling*, *rapid elasticity* and *measured service*. Indeed, the popularity of Cloud computing is defined by its characteristics to offer flexible and fast resources and an economy of scale in which a user pays for Cloud service on consumption.

There are a large number of different services, such as servers, storage, databases, analytics and so on, that can be distinguished into three main types:

1. *Infrastructure as a Service (IaaS)*: it is the most basic category of Cloud computing, with which the customer can rent IT infrastructures, such as servers and virtual machines, storage, networks and operating systems.
2. *Platform as a Service (PaaS)*: with PaaS, users can focus on the deployment of applications instead of caring about the underlying infrastructure. Indeed, these services supply an on-demand environment for developing, testing, delivering and managing software applications.
3. *Software as a Service (SaaS)*: this kind of service consists of a complete product that is run and managed by the service provider. Usually, these applications are feasible through a program interface or using web browsers.

2.3.2 Microsoft Azure

Microsoft Azure[1] is a Cloud platform owned by Microsoft. According to Microsoft [Azu, c], today Azure platform offers more than 200 products and services. Azure provides three types of Cloud computing [Collier and Shahan, 2015]: SaaS such as Office 365, PaaS like Azure Cloud Services, and IaaS which allows the creation of Virtual Machines. Azure offers services that support a variety of programming languages, including popular ones like Python and JavaScript and, additionally, it integrates with a large number of widely used tools (such as Visual Studio). Users leveraging Azure can develop new applications entirely in the Cloud environment or use it to run existing applications.

According to the Microsoft directory [Azu, b], Azure products can be classified into

[1]Microsoft Azure official website: `https://azure.microsoft.com/it-it`.

21 categories: AI+Machine Learning, Analytics, Compute, Containers, Databases, Developer tools, DevOps, Hybrid + multi-cloud, Identity, Integration, Internet of Things, Management and governance, Media, Migration, Mixed Reality, Mobile, Networking. security, Storage, Virtual desktop infrastructure, and Web. As illustrated in the next chapters, in this project are used an AI + Machine Learning service, Azure Open AI, two analytic services, Azure Data Factory and Azure Databricks, and Azure Data Lake storage as storage.

Azure OpenAI

Azure Open AI [Azu, a] is classified as an AI + Machine Learning service and provides REST API access to Open AI models. OpenAI (see Subsection 3.3.2) is an American company specializing in Artificial Intelligence and the development of Large Language Models, which are detailed further in Section 3.2. With Azure OpenAI, users can integrate OpenAI models with other Azure Cloud services to develop applications that harness Artificial Intelligence directly in a Cloud environment. Azure OpenAI integrates the advantages of the Microsoft Cloud platform with the latest Artificial Intelligence models developed by OpenAI, allowing direct interaction with other components of the Microsoft data platform used for data storage, analysis and preparation in the context of the experiments. Azure OpenAI consists of four main components:

1. pre-trained generative AI models;
2. customization options with the possibility to fine-tune AI models with personalized data;
3. in-built tools for identifying and addressing potential harmful uses, promoting responsible AI implementation;
4. robust enterprise security featuring role-based access control (RBAC) and private network functionalities.

Azure Data Factory

Azure Data Factory[Azu, d] is a serverless and code-free data integration service provided by Microsoft Azure. This service enables the creation of data-driven workflows to orchestrate the movement and transformation of data on a large scale. The platform is capable of orchestrating a vast amount of raw data through complex hybrid extract-transform-load (ETL), extract-load-transform (ELT) and data integration projects. It facilitates communication with various data sources simply and immediately, leveraging the features of the Azure Cloud. As a code-free platform, data transformations can be implemented by anyone through a user-friendly interface. Azure Data Factory manages data transformation at the Spark scale, without requiring the user to set up or fine-tune a Spark cluster.

Azure Databricks

Azure Databricks [Azu, e] is a platform offered as a SaaS (Software as a Service) by Azure, enabling the utilization of Apache Spark, an open-source, in-memory data processing engine for AI and Machine Learning applications. It serves as a unified, open analytics platform for building, deploying, sharing and maintaining enterprise-grade data, analytics and AI solutions at scale. This platform is a fully managed notebook-oriented version of the open-source Apache Spark analytics and data processing engine, tailored specifically for substantial data processing needs. Data scientists benefit from the platform's built-in core API, supporting key languages such as Python, R, SQL, Java, and Scala, the primary language of Apache Spark. Noteworthy characteristics of Databricks encompass simplicity, openness and compatibility with multi-cloud environments. Azure Databricks combines the power of Apache Spark with a user-friendly, collaborative and scalable platform, making it a versatile solution for organizations aiming to harness the potential of big data analytics and AI at scale.

Azure Data Lake storage

Azure Storage is designed to store and manage large volumes of data in various forms, such as structured data, unstructured data and binary data. It is the overarching service that provides Cloud storage solutions, such as Blob Storage or Data Lake Storage, which are the most common ways to store unstructured data in Azure. Blob storage is a service designed for storing and managing unstructured data, typically in the form of files or objects. Azure Data Lake is designed for big data analytics and data warehousing scenarios. It is optimized for handling large volumes of structured and unstructured data, making it ideal for data lakes needed to store, analyze, and process massive datasets.

Chapter 3

STATE OF THE ART ON LARGE LANGUAGE MODELS

This chapter provides an overview of the current state of Large Language Models (LLMs). Section 3.1 elucidates the relationship between Natural Language Processing and Language Modeling, underscoring its pivotal role in crafting computational models capable of handling natural language in a way that mirrors human expression. Within the diverse landscape of Natural Language Processing tasks, Subsection 3.1.1 directs attention to the realm of emotion detection, a focus pertinent to the current use case. Section 3.2 delves into the definition and architectural description of Large Language Models, focusing on the Transformer architecture in Subsection 3.2.1. The succeeding Section 3.3 elucidates the relationship between Natural Language Processing and Generative AI, highlighting the potency of Generative Large Language Models, explicated in Subsection 3.3.1. This is followed by Subsection 3.3.2, which positions OpenAI as one of the contemporary company leaders in Large Language Model production. Section 3.4 outlines the key elements of prompt engineering concerning generative LLMs, with a particular emphasis on certain techniques in Subsection 3.4.1.

3.1 Natural Language Processing and Language Modeling

According to the definition proposed by Elizabeth D. Liddy [Liddy, 2001], Natural Language Processing (NLP) is a comprehensive set of computationally driven techniques designed to analyze and represent naturally occurring texts at various linguistic levels. A fundamental objective within the realm of NLP tasks is to equip computers with the capability to comprehend the entirety of meaning conveyed through language inputs whether in textual or vocal formats and process them with a level of sophistication similar to human understanding. NLP is intrinsically related to the expansive realm of Artificial Intelligence, aiming to empower machines with the capability to effectively interpret human natural language. This intricate process entails the seamless integration of various elements, including linguistic theories and statistical methodologies, as well as sophisticated Machine Learning and Deep Learning models.

The field of NLP presents numerous challenges. Examples include text classification, where algorithms strive to categorize textual content into predefined classes or categories, named entities recognition (NER), which aims to identify entities like people, places or locations, topic recognition, to identify different topics discussed in a given document, or sentiment analysis and emotion detection, focusing on discerning the emotional tone or attitude expressed in a piece of text. Section 3.1.1 explains the emotion detection task in detail, which is particularly relevant to this case study.

In this complex landscape of Natural Language Processing, Language Modeling is pivotal in empowering machines to handle natural language input and output. Language Modeling seeks to capture the generative likelihood of sequences of words by predicting the probabilities associated with upcoming or missing tokens [Zhao et al., 2023]. It involves the development of computational frameworks that enable devices to understand and generate human-like language. This capability becomes particularly significant in addressing the challenges posed by NLP tasks, where reducing the complexity of human language to more formal structures is essential. Language Models contribute to all the previously listed endeavors by enhancing machines comprehension of language structures and context. These models, in turn, prove indispensable in real-world business applications, facilitating tasks like customer satisfaction analysis or the development of advanced conversational interfaces such as chatbots and voicebots.

Over the past two decades, Language Modeling has gone through four different phases [Hadi et al., 2023]. In its initial stages, Language Modeling was carried out using Statistical Language Models (SLMs), like n-gram models, capable of predicting the probability of one word following another based on the frequency of occurrences of previous n-grams of a word. Subsequently, statistical models were surpassed with the advent of Neural Language Models (NLMs), like Recurrent Neural Networks (RNNs), which relied on Neural Networks to calculate the probability distribution of the next word in a sequence given the preceding words in the sequence. The next stage of Language Modeling saw the introduction of Pre-trained Language Models (PLMs), which use Neural Networks to vectorize words to consider their context. Following this phase, there was the development of PLMs with an extremely high number of parameters, the so-called Large Language Models (LLMs). These models, which are described more in detail in Section 3.2, are trained on large amounts of data and are capable of performing tasks at a level comparable to humans. LLMs reflect the state-of-the-art in the NLP landscape, representing the cutting edge in language understanding with unprecedented capabilities in processing and generating human-like text.

Language Modeling extends beyond mere input and output handling, serving as a cornerstone for the development of AI systems that can navigate the intricacies of human communication with increased fluency and precision. With the advanced architectures of LLMs and their extensive training on enormous datasets, these models are highly proficient in various NLP tasks and play a crucial role in driving innovation and pushing the boundaries of what is achievable in the field of natural language understanding.

3.1.1 Emotion Detection

Emotion detection is an NLP task whose aim is to identify emotions within a text, classifying them more or less granularly based on emotion models. Quoting the definition proposed by Kao et al. [Kao et al., 2009], the task can be mathematically expressed as $r : A \times T \rightarrow E$, where T represents the text and A is the author. The variable r denotes the relationship between the author and the written text, often expressing emotions, E.

Despite the seemingly straightforward nature of the problem, determining the appropriate relation under which an author can be significantly associated with their written texts to discern emotions can be challenging. Indeed, the complexity of the task lies in its inherently semantic nature. To ascertain the emotion of someone writing a text, it is necessary to have a comprehensive understanding of the context in which certain words are placed, capturing the associated tone. Additionally, the introduction of various ambiguities, slang, and terminologies over time adds to the challenge of emotion detection from text. Furthermore, emotion detection goes beyond identifying primary psychological conditions (happy, sad, anger), extending to multi-scale classifications depending on the emotion model [Nandwani and Verma, 2021]. In the contemporary landscape, the emotion detection task can be addressed using Machine Learning and Deep Learning tools trained on annotated datasets or through prompting with generative LLMs (see Section 3.4).

The growing popularity of emotion detection is closely tied to the widespread availability of network resources, particularly those from social networks. In the contemporary landscape, companies have recognized the value of employing emotion detection as a strategic tool to detect nuanced insights into the dynamic landscape of customer experiences. Moreover, the integration of emotion detection into business analyses facilitates a more holistic comprehension of customer satisfaction and engagement. Since it is possible to derive actionable insights from the emotional cues embedded in communication channels, emotion detection has become an integral component of customer-centric strategies that prioritize not only what customers say but also how they feel about products and services.

3.2 Large Language Models

A Large Language Model, often abbreviated as LLM, is an Artificial Intelligence system that leverages advanced statistical models and Deep Learning techniques to process textual data written in human natural language [Hadi et al., 2023]. The architecture of LLMs is based on the Transformer (refer to the next Subsection 3.2.1), containing an extensive number of parameters, often in the order of billions. These models undergo a pre-training phase exposed to a massive amount of textual data from various sources. During this phase, they learn to predict the next word in a sequence based on preceding ones, recognizing language patterns that enable them to comprehend and generate text. Notable applications of LLMs include language translation, text summarization, question answering, content generation and more.

3.2.1 The Transformer

The Transformer is an architecture that relies on the parallel multi-head attention mechanism designed to handle text sequences [Vaswani et al., 2017].

At its core, the Transformer employs a self-attention mechanism, a fundamental component that empowers the model to discern and capture intricate relationships between individual words and other words within the same sentence. Indeed, what makes LLMs able to generate text in a human-like way is the fact that the Transformer processes a whole sequence at once, not analyzing only individual words but taking into consideration the context of a sentence. For every token of the sentence, the Self-attention mechanism captures the other ones that are most important to understand its meaning. Being able to understand the context of a text by analyzing a whole sentence at once allows the model to recognize the semantics of words that have multiple meanings based on the other ones in the sentence [Vaswani et al., 2017]. Expanding upon the concept of self-attention, multi-head attention is an enhancement that involves simultaneously considering multiple distinct embeddings, enabling the model to focus on various positions within a given sequence.

The Transformer architecture, which is shown in Figure 3.1, is mainly made by an Encoder component and a Decoder component connected. The Encoding component is made of six identical stacks of encoders one on top of the other, each one consisting of a self-attention layer connected with a feed-forward Neural Network. The decoding component is constructed in the same way as the Encoding one, with six decoding stacks, each one composed of a self-attention layer, an additional encoder-decoder attention layer to focus better on relevant information of the input sentence, and the same feed-forward Neural Network on top. Both in the encoder and in the decoder component, between each layer inside each stack, there is a normalization layer. Concerning the functioning of the model, as described by Vaswani et al. [Vaswani et al., 2017], each word in the sentence is first vectorized and transformed into an embedding. To account for the position of words in a sentence, a positional vector is added to each embedding. The resulting list of vectors is given as input into the encoder and passed through all its six layers. At this point, the output of the entire encoder component is transformed into a set of attention vectors used in each decoder stack in their encoder-decoder attention layer. These vectors are passed to the decoder component and traverse all six layers, producing a vector of floats. The vector is then passed to a linear layer followed by a Softmax layer. In the linear layer, the vector output by the decoder is projected

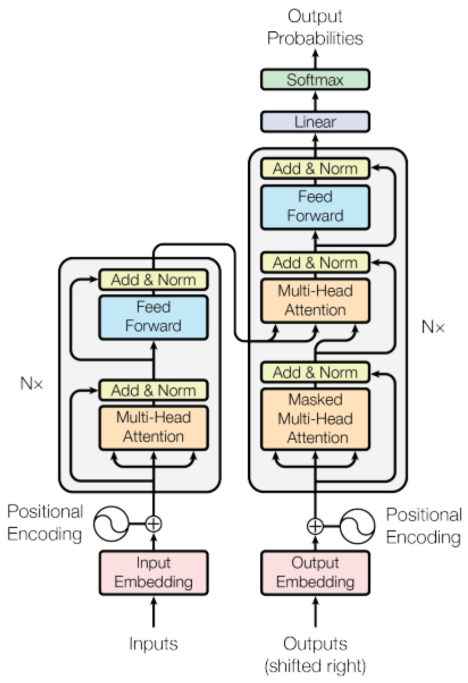


Figure 3.1: Transformer Architecture from Vaswani et al. [Vaswani et al., 2017].

into a logits vector, which is then transformed into a vector composed of probabilities by the Softmax layer. To convert the numbers into words, the word connected to the cell with the highest probability is selected as the output.

3.3 Natural Language Processing and Generative AI

Generative AI is a field of Artificial Intelligence characterized by the generation of high-quality, human-like content such as texts or images generated as answers to a given instruction written in natural language, known as prompt. The term Generative AI is usually connected with complex Large Language Models able to generate previously unseen information [García-Peñalvo and Vázquez-Ingelmo, 2023], reproducing the characteristics of the data on which they were trained.

In contemporary times, the relationship between Natural Language Processing and Generative AI is particularly strong concerning Natural Language Generation. Natural Language Generation consists of both converting input texts into new symbol sequences (e.g. text summarization and machine translation) and generating text from scratch to match input description (e.g. writing a text based on some instructions) [Yang et al., 2023]. Generative AI requires human guidance through written instructions, known as prompts. The definition of the prompt is crucial for utilizing generative models in the most efficient manner possible and is the subject of study in prompt engineering, which is more comprehensively discussed in Section 3.4.

3.3.1 Generative Large Language Models

Building upon the Transformer architecture presented in previous Subsection 3.2.1, LLMs can be classified into three families: one composed of encoder-decoder architecture, such as T5 [Raffel et al., 2020], one made of encoder-only models, such as BERT [Devlin et al., 2018], and one formed by decoder-only models, which nowadays is dominating the development of LLMs. In the current study, we focus on decoder-only models referred to as Generative Large Language Models. More specifically, we concentrate on Generative Pre-trained Transformer (GPT) models, trained by predicting the next word in a sequence based on the preceding words [Yang et al., 2023].

Released by OpenAI (see Section 3.3.2), Generative Pre-trained Transformer models are a series of Large Language Models that utilize Deep Learning techniques to comprehend and generate human-like text. GPT Language Models gained significant popularity with the release of ChatGPT, which adapts LLMs from the GPT series for dialogue, demonstrating remarkable conversational abilities with humans. GPT models have been presented in Radford et al. [Radford et al., 2018], where generative pre-training has been defined as "the ability to train Language Models with unlabeled data and achieve accurate prediction". These models have been shown to be able to deal with several Natural Language Processing tasks in zero-shot or few-shot settings [Brown et al., 2020]. Indeed, thanks to their exposure to a broad spectrum of linguistic patterns and contexts during pre-training, these models acquire a generalized understanding of language, allowing them to handle various NLP tasks out of the box. Moreover, these models can be fine-tuned to carry out downstream tasks with even greater precision in specific domains. Using domain-specific data, model knowledge is leveraged to better suit the characteristics of the target domain. This capability is particularly valuable in scenarios where a precise and domain-specific understanding of language is required, such as in professional applications, research or industry-specific analyses.

As shown in Figure 3.2, GPT models have a decoder-only architecture built on Transformers, in which the decoder component is the same as described in Subsection 3.2.1. They are composed of 12 decoder blocks, followed by a linear layer and a Softmax layer, as for the Transformer model previously described. As far as the function is concerned, the input is given to the Transformer in the form of vectorized textual prompts, while the output depends on the task for which the model is used. For tasks like the next word or sentence prediction, the returned output will be the probability distribution of the next token/word following the prompt. The choice of the returned word is based on the self-attention mechanism, which has determined a higher attention on the identified word.

The initial models in the GPT series were GPT-1 [Radford et al., 2018], released in 2018, followed by GPT-2 [Radford et al., 2019], announced in 2019, which were significantly surpassed by 2020 GPT-3 [Brown et al., 2020] on several aspects. GPT-3, trained with 175 billion parameters, excels in zero-shot learning, performing tasks for which it hasn't been explicitly trained. With few-shot learning, GPT-3 rapidly adapts to new tasks and domains with minimal training. Subsequently, the GPT-3.5 series models were developed, relying on Reinforcement Learning with Human Feedback. In March 2023, OpenAI announced GPT-4 [OpenAI, 2023a], the latest GPT-like model with multimodal capabilities to process both textual and visual inputs.

Despite the remarkable accomplishments of Large Language Models as versatile

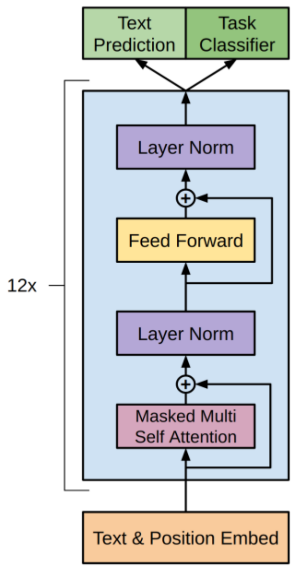


Figure 3.2: GPT Transformer Architecture from Radford et al. [Radford et al., 2018].

problem-solving agents, they still have problems in tasks that demand extensive knowledge and intricate reasoning. Moreover, among the most significant issues are hallucinations, which refer to the generation of nonsensical, inconsistent or incorrect content that deviates from reality due to a lack of coherence with the input instructions [Huang et al., 2023].

3.3.2 Open AI

OpenAI [OpenAI, 2023b] is an American AI research and deployment company consisting of both for-profit and non-profit entities whose research is mainly connected with generative models for both text and visual content. OpenAI was founded in December 2015 by Elon Musk, Sam Altman, Greg Brockman, Ilya Sutskever, John Schulman and Wojciech Zaremba. Since 2020, when they announced GPT-3 (see Section 3.3.1), OpenAI has significantly increased its fame, becoming one of the most widely internationally discussed names with the launch, in 2022, of Chat-GPT (see Section 3.3.1). OpenAI has adopted the practice of providing access to their research findings and, in some cases, releasing models and code to the public. The OpenAI API is powered by different sets of models with different capabilities and price points, which can also be customized for specific use cases with fine-tuning. Among the available ones, some of the most popular models are:

- GPT-models: a set of models able to understand and generate natural language. The last models are GPT-3 and its improvement GPT-3.5 and GPT-4 and GPT-4 Turbo;
- DALL-E: for generating and editing images;
- Whisper: for converting audio into text.

3.4 Prompt engeneering

As illustrated in the previous Subsection 3.3.1, Generative LLMs require human instruction written in natural language to perform a task. This instruction is known as prompt and the techniques used to craft it can significantly impact the performance of the LLM. Prompt engineering is a discipline that has gained prominence in recent times, focusing on a wide range of techniques and competencies to enhance the interaction and development of LLMs. Consistent with the experiments and analyses reported in the subsequent sections, the following considerations on prompt engineering are exclusively related to the generation of textual content.

As explained by Liu et al. [Liu et al., 2023], prompt-based learning, commonly known as in-context learning, diverges from conventional supervised learning. In contrast to a traditional learning pipeline where the model is asked to generate an output y in response to an input x, the input x is presented to the model as a textual prompt *x'* containing missing components. These empty slots are probabilistically filled by the model, producing a string $\hat{x}$ from which the final output y can be derived. Through this form of learning, the LLM is capable of performing a large number of tasks with minimal or no labeled data, relying only on the data on which it was trained. This approach addresses a major challenge of supervised learning, namely the need for vast datasets of labeled examples to train the model. The prediction of the output y involves primarily three steps, utilizing the probability of the input x itself [Liu et al., 2023]:

1. Prompt addition: this step consists of applying to the input x a function f in which it is applied a template. This template is a textual string composed of an input slot [X], that is filled with the input x, and an answer slot [Z]. for an interrelated generated answer z. This phase returns $x' = f_{\text{prompt}}(x)$.

2. Answer search: in this subsequent phase, a set of permissible values for z, named Z, is defined. This set can be constrained to a defined set of classes or consists of the entirety of the language. Subsequently, different values of z from the set are inserted into the [Z] slot of the template x', and the most probable z within the prompt, denoted as $\hat{z}$, is determined by a pre-trained LM based on a function (such as an ArgMax).

3. Answer mapping: finally, the most probable $\hat{z}$ is transformed into the output $\hat{y}$.

Through prompts it is possible to ask the model to perform a wide variety of tasks such as writing contents in different styles, summarizing texts, structuring data, retrieving information or solving NLP tasks, or detecting customized features in a text, as it is presented in this study.

3.4.1 Prompting techniques

With the recent proliferation of increasingly powerful generative models, numerous studies have been conducted based on various prompting techniques. Below, some of the most established and widely used techniques are outlined:

1. *Few-shot prompting*: introduced in the foundational paper on GPT-3 by Brown et al. [Brown et al., 2020], few-shot prompting involves presenting the model with a small set of input and output examples within the prompt before instructing

it to perform a particular task. Few-shot prompting has become a prevalent technique in interacting with LLMs through prompts, allowing for improved model understanding and performance across a range of tasks.

2. *Chain-of-Thought Prompting (CoT)*: this technique was introduced by Wei et al. [Wei et al., 2022] and is based on inducing reasoning in LLMs by breaking down the task into intermediate steps. Kojima et al. [Kojima et al.,] extend the zero-shot chain of thoughts by adding expressions within the prompt to stimulate reasoning, such as *"Let's think step by step"*.

3. *Self-Consistency*: Self-Consistency, presented by Wang et al.[Wang et al., 2022], consists of diversifying the reasoning paths by sampling instead of solely relying on a greedy approach and, subsequently, identifying the most consistent answer by marginalizing the sampled reasoning paths. This technique is based on the intuition that complex reasoning problems often have multiple valid approaches, all leading to the same correct answer.

4. *Role Prompting*: generative models have been shown to gain particular benefits from specifying a role in the prompt that they have to play in solving the proposed task [Xu et al., 2023]. This could be done, for example, by including directives at the beginning of the instruction such as *"Act as a lawyer and analyze texts in the legal field"*.

In today's context, the study of prompting techniques is undergoing continuous evolution. This is particularly evident as the composition of instructions in natural language not only facilitates but also broadens the potential for achieving outstanding performance LLMs, even in the context of more intricate tasks. As we delve deeper into the intricacies of natural language, we unlock the potential to refine and expand the range of tasks that these models can proficiently handle.

Chapter 4

SOLUTION DESIGN

The present chapter outlines the design of the proposed solution for the given use case. Section 4.1 provides a comprehensive description of the case study, which aims to define some interesting features to be extracted from chats with the customer service of the company using LLMs. Section 4.2 contains a description of the data and depicts the processing that data undergoes to be cleaned and prepared for subsequent analysis. Data supplied by the company is a set of chats extracted from the company website written in Italian. Section 4.3 provides a detailed overview of the data, consisting of graphs and statistics that explore various interesting facets. Section 4.4 outlines the features identified to extract as the final output of the process, serving as intriguing inputs for future analyses within a broader business context. In the end, Section 4.5 describes the architectural framework employed in the solution, constructed on a cloud infrastructure using Microsoft Azure services.

4.1 Case study

The case study centered on developing a chat analysis system incorporating contemporary cloud-based advanced analysis solutions and cutting-edge Large Language Models. The company provides an online chat support service for customers accessible from their website, intended to help customers address concerns and acquire a range of information related to the company's gaming platform. The primary objective was to delineate a collection of features pertinent to the business context for each extracted conversation from the website and assign them a value, leveraging the adaptability of generative LLMs. This process aims not only to conduct a qualitative analysis of the specified set of conversations but also to render them usable as input for future evaluative systems, as elaborated further in Section 6.2. In this phase, the case study was designed as an experiment on a sample of chats, striving to define the essential characteristics of the architecture. The prospective client company's purpose is to integrate this designed analysis architecture based on Large Language Models into their internal Machine Learning framework, extracting real-time features for every user interaction with the chat service.

Figure 4.1 illustrates the complete workflow of the use case. In the initial step, data were migrated from the client's storage to a data lake within the use case architecture. Subsequently, a meticulous analysis was conducted to identify errors, missing values, and null values, addressing them where feasible. This phase was complemented by a statistical analysis of the sample, offering a comprehensive understanding of the data and contributing to the definition of features for extraction. Particularly crucial in this exploratory and defining phase was the collaboration with the client company, where a combination of technical expertise and domain knowledge was essential to define the desired output and align it with potential business objectives.

Once the output was defined, an extensive experimentation phase was conducted, involving the crafting of prompts as instructions to obtain the desired output. At this stage, the data underwent preparation in both content and format to serve as input for the chosen Large Language Model. The selection of the Language Model was carried out by striking a balance between costs, execution times, and performance, as detailed in Section 5.1, leading to the choice of GPT-4-Turbo by OpenAI (refer to Section

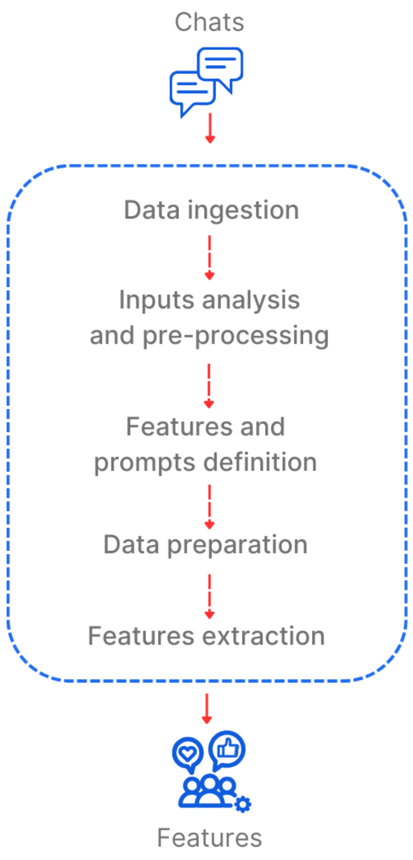
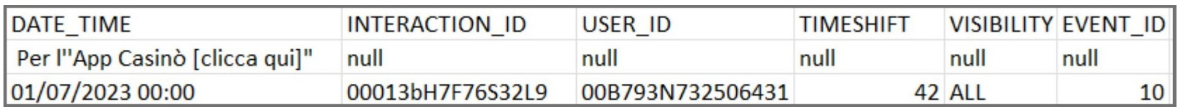

Chats
Data ingestion
Inputs analysis and pre-processing
Features and prompts definition
Data preparation
Features extraction
Features

Figure 4.1: Case study workflow.

3.3.1). As previously explained, the essence of the use case lies in the extraction of features for each conversation within the sample. These features subsequently undergo a comprehensive analysis to derive preliminary insights and explore their potential applicability for other use cases to detect their broader utility and versatility.

4.2 Dataset description and data preprocessing

The data employed for subsequent analysis is provided by the client company and includes a compilation of chat conversations written in Italian. These conversations are stored in a CSV file where each row contains a message related to a conversation, reconstructable in its entirety through a unique identifier for each chat. The data is characterized by various key attributes, including temporal information, user details and identifiers, crucial for reconstructing the interaction or providing customer infor-



DATE_TIME	INTERACTION_ID	USER_ID	TIMESHIFT	VISIBILITY	EVENT_ID
Per l"App Casinò [clicca qui]"	null	null	null	null	null
01/07/2023 00:00	00013bH7F76S32L9	00B793N732506431	42	ALL	10

Figure 4.2: Example of a record considered as an error.

mation. The original dataset comprises 6.400.619 rows and 14 columns. Table 4.1 provides specific information about the data type, description, and potential values for each of the 14 attributes.

The first operation conducted to explore the data consisted of identifying missing values and any potential errors within the dataset. A total of 964.921 rows were classified as errors, encompassing:

- completely empty rows;
- rows in which a single column was set. An illustrative example of this error category is provided in Figure 4.2, where a record considered as an error in the first row is compared to a correct one in the second row.

The identified errors were removed from the dataset and stored in a separate CSV file. The total number of rows in the new corrected dataset is 5.435.698.

At this stage, the data from the cleaned dataset underwent preprocessing to address missing values where feasible. Specifically, it was observed that certain attributes were populated based on others only at their initial occurrence. For example, the user's nickname, linked to the user ID, was explicitly specified only at the first instance of the ID, remaining null in subsequent occurrences of the same ID. This pattern holds for attributes such as *Person ID*, *Usernick*, *Usertype*, *Protocol type* and *Client version*, which were subsequently populated, leading to a significant reduction in the number of missing values. Table 4.3 displays the final count of missing values for each attribute compared to the initial number detected. Through the analyses conducted on the data and outlined in Section 4.3, it was clarified that some of the null values in the data are not real missing values but are determined by the nature of the attribute in which

Attribute	Data Type	Description	Possible Values
DATE_TIME	Timestamp	Start time of the chat	Date and time
INTERACTION_ID	String	Unique identifier for the interaction	Alphanumeric ID
USER_ID	String	Unique identifier for the user	Alphanumeric ID
TIMESHIFT	Integer	Time interval in seconds since joining the chat	1, 2, 3, . . . , n
VISIBILITY	String	Visibility setting for the chat	ALL
EVENT_ID	String	Sequential number assigned to each chat record	1, 2, 3, . . . , n
PERSON_ID	String	Nickname of the agent	Agent nickname
USERNICK	String	Nickname of the user	System, Live chat, User or Agent nickname
USERTYPE	String	Type of user	Client, External, Agent
PROTOCOLTYPE	String	Type of the protocol used	FLEX (client), ESP (routing strategy), BASIC (agent)
TZOFFSET	Integer	Time zone offset	0
MSGTEXT	String	Text of each chat message	Text message
ASKERID	String	ID of the user closing the chat	ID of the user closing the chat
REASON	String	Reason for chat closure	Left, Left with request to close forcibly

Table 4.1: Dataset Attributes Overview.

they are present. All the subsequent analyses were conducted on the final dataset, which was filled and cleaned and consists of 5.435.698 rows. Table 4.2 compares the

DATASET	NUMBER OF ROWS
Initial dataset	6.400.619
Error dataset	964.921
Final cleaned and filled dataset	5.435.698

Table 4.2: Number of rows for the three different datasets.

row numbers in the initial dataset, in the one where errors are stored and in the final dataset.

For further details, the code employed for data preprocessing is reported in A.1.

4.3 Data exploration

Following the pre-processing phase, the data has undergone statistical exploration. All statistical analyses were conducted using SparkSQL and PySpark in a Databricks notebook, given that Databricks played a central role in data processing within the use case architecture, as detailed in Section 4.5. The dataset comprises 265.592 distinct conversations and 5.435.698 messages, with each row representing an individual message.

First statistical analyses were conducted with a focus on users. The company's website initiates the customer's first interaction through an automated live chat, directing the conversation toward predefined potential issues. If the live chat assistance is not sufficient, the customer has the option to communicate with a human operator. Within this framework, messages can be attributed to three user types: *Agent*, representing operators and the Live chat, *Client*, the company's customers, and *External*, the automatic chat open/close system. Figure 4.3 illustrates the percentage distribution of messages for each user type relative to the total, with 2.278.160 messages written by clients, 2.621.091 by agents, and 536.447 by the automatic system.

Considering that messages labeled as *Agent* comprise both automated Live Chat

Attribute	**Starting Missing Values**	**Final Missing Values**
DATE_TIME	502	0
INTERACTION_ID	960.625	0
USER_ID	964.801	0
TIMESHIFT	964.845	0
VISIBILITY	964.888	0
EVENT_ID	964.894	0
PERSON_ID	6.288.706	4.541.395
USERNICK	5.512.940	0
USERTYPE	5.512.953	0
PROTOCOLTYPE	5.512.963	0
TZOFFSET	964.921	0
MSGTEXT	2.481.716	1.517.660
ASKERID	5.808.687	4.843.766
REASON	5.771.673	4.806.752

Table 4.3: Comparison of initial and final missing values for each attribute.

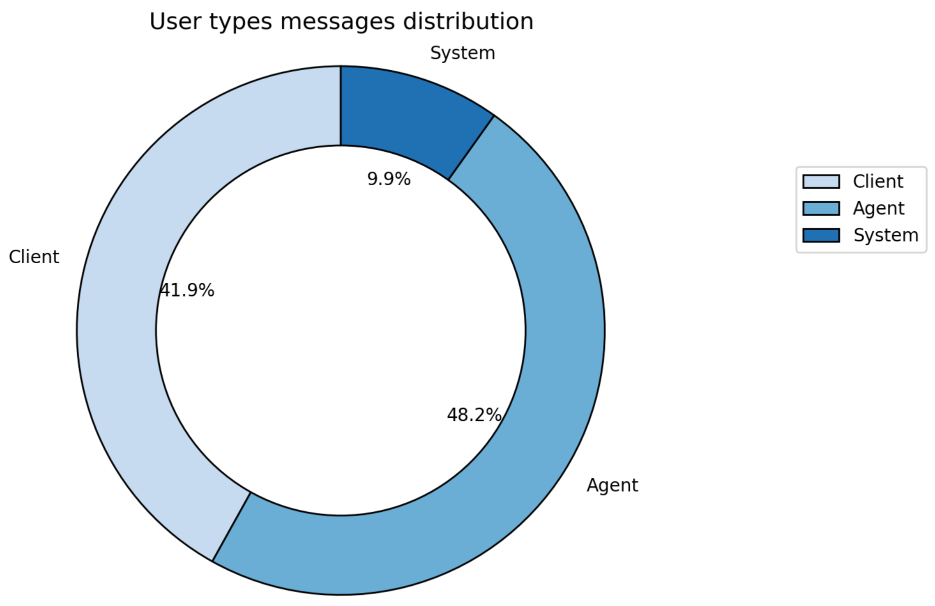


Figure 4.3: User types distribution in messages.

messages and those authored by human operators, a more refined analysis was conducted. Leveraging the *Usernick* attribute, indicating whether the agent is the live chat, it was identified that out of 2.621.091 agent messages, 894.303 were authored by human operators and 1.726.788 by the live chat. The percentages of messages by human operators or by Live chat are shown in Figure 4.4. Therefore, as shown in Figure 4.5 out of the total messages, 3.172.463 were authored by humans, so by customers and customer care agents, whereas 2.263.230 were generated either by the automatic live chat or by the system, and have a non-human origin.

Additional relevant information pertains to returning customers, individuals who have engaged with the service on multiple occasions. The dataset comprises a total of 102.395 distinct customers, of which 46.198 have initiated multiple connections. Customer-type percentages are reported in Figure 4.6.

Regarding the null messages discussed in the previous Section 4.2, out of a total of 1.517.660 messages flagged as null in Table 4.3, only 628.946 can be actually classified as true nulls, as shown in Figure 4.7. This clarification is necessary because the dataset is organized in a way that assigns null values to the message text in all rows where the

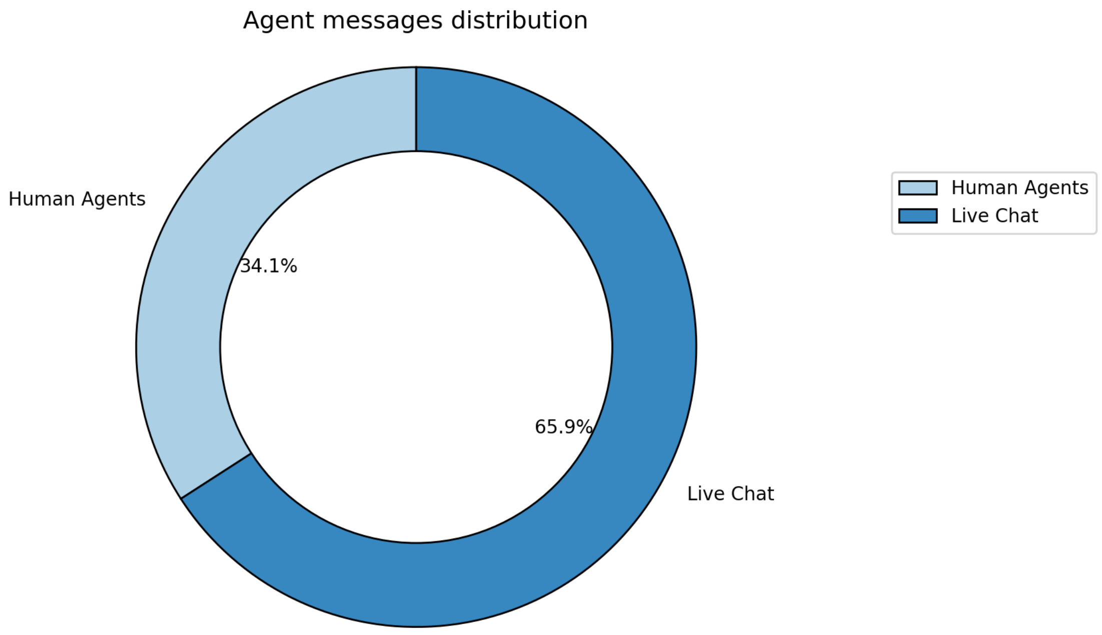


Figure 4.4: Agent messages distribution.

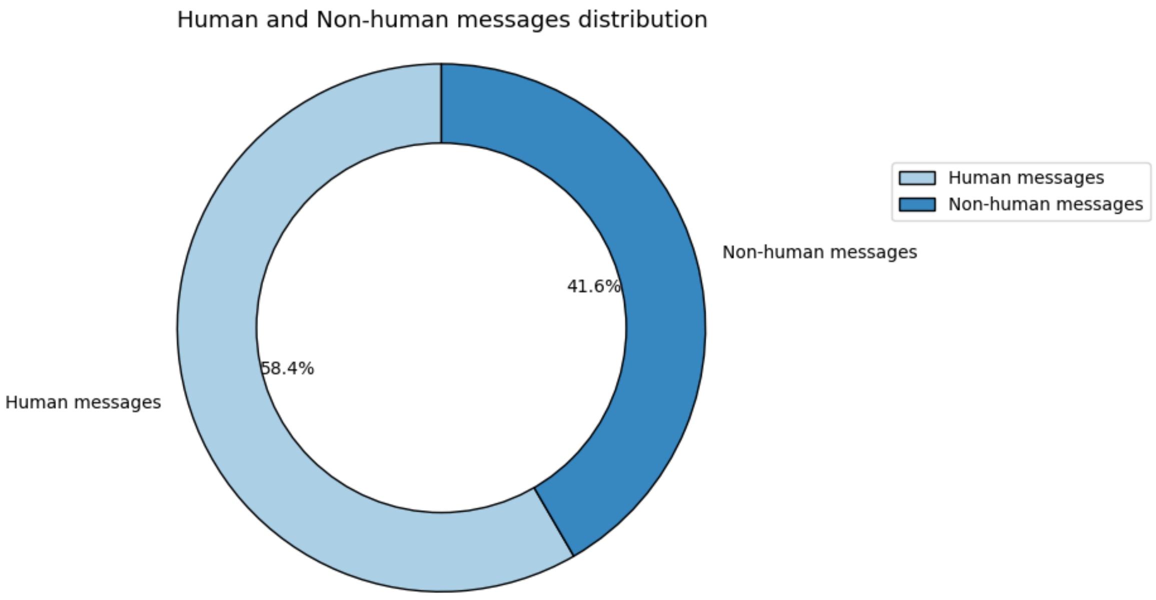


Figure 4.5: Human and non-human messages distribution.

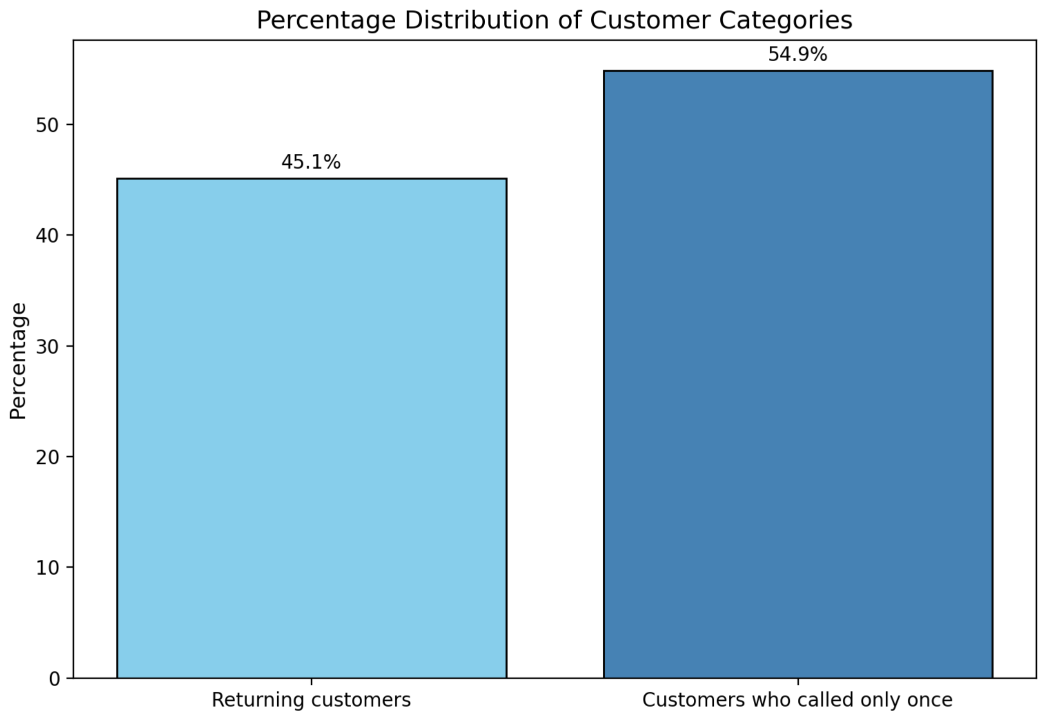


Figure 4.6: Customers types distribution.

chat closure request is recorded.

To facilitate a comprehensive analysis and differentiate between automated and human-driven interactions, the count includes conversations concluded without any human operator involvement. These amount to 154.062 cases out of 265.592, signifying instances where conversations are either proactively closed or resolved independently by the automated live chat without requiring human intervention.

Afterward, analyses were conducted on the chat's length, considering both its duration and the number of messages in the interaction. Duration statistics are presented in Table 4.4. An outlier was identified as the maximum chat duration in minutes, representing a scenario where the customer initiated the chat but neither entered any text nor closed the conversation, leading to automatic closure by the system due to reaching the maximum duration. It was observed that the majority of chats with an exceptionally long duration were similar cases. Despite this, these cases were retained within the sample. This choice stems from the need for the analytical framework to accommodate various chat interactions on the website, making it potentially valuable

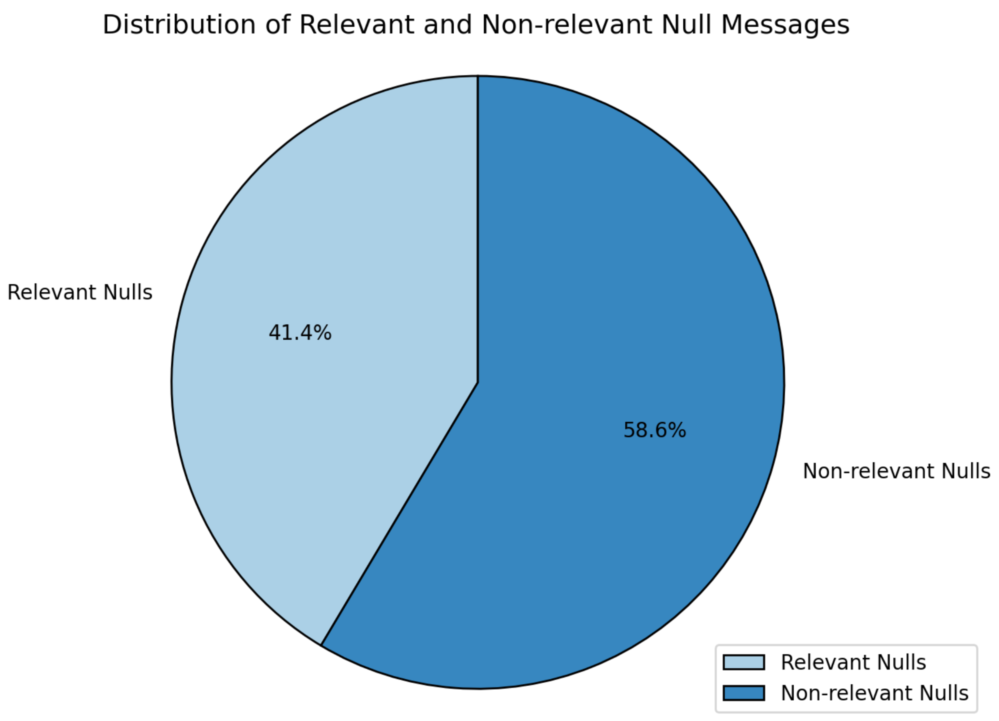


Figure 4.7: Null messages distribution.

	Duration		
	Avg	Max	Min
In minutes	4.16	9207	0.02
In messages	20.47	216	2

Table 4.4: Duration of conversations in terms of messages and minutes.

to study the system's behavior in such cases. For both the distribution in terms of the number of messages and the duration in minutes, the standard deviation was calculated. It amounts to 20,38 for the duration in minutes and 12,35 for the number of messages. Regarding chat duration, a standard deviation of 20,38 minutes indicates considerable variability, with some chats lasting significantly longer or shorter than the average. Similarly, a standard deviation of 12,35 for the number of messages suggests variability in the engagement levels of different chat interactions. Figures 4.8 and 4.9 depict the distributions of chat durations, with the maximum set at 40 for better representation due to the limited number of data points with higher values.

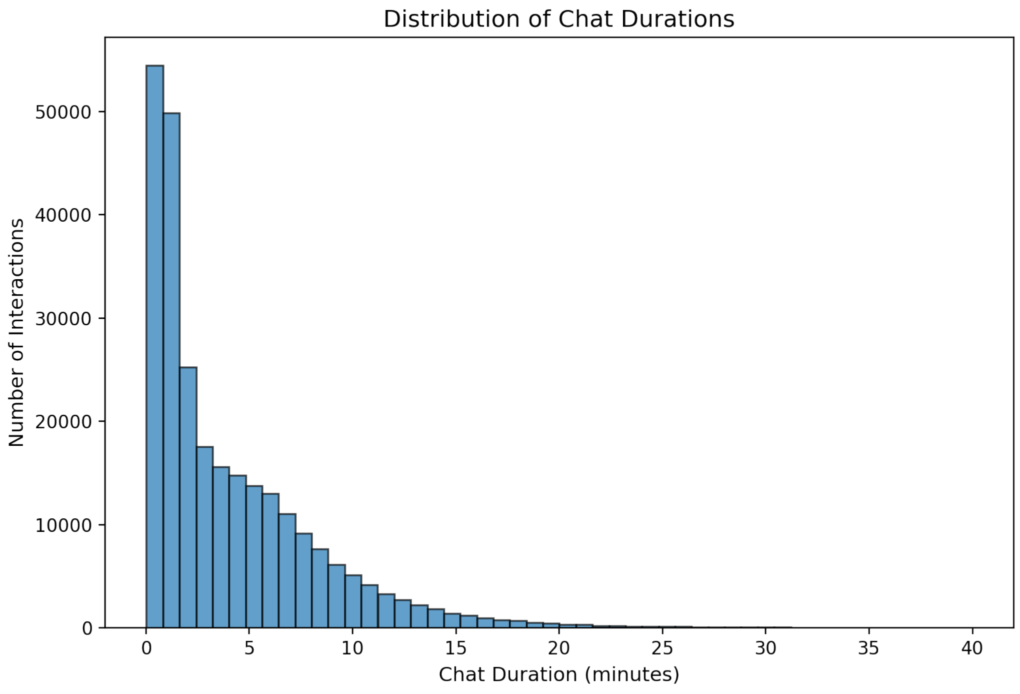


Figure 4.8: Chat durations in minutes.

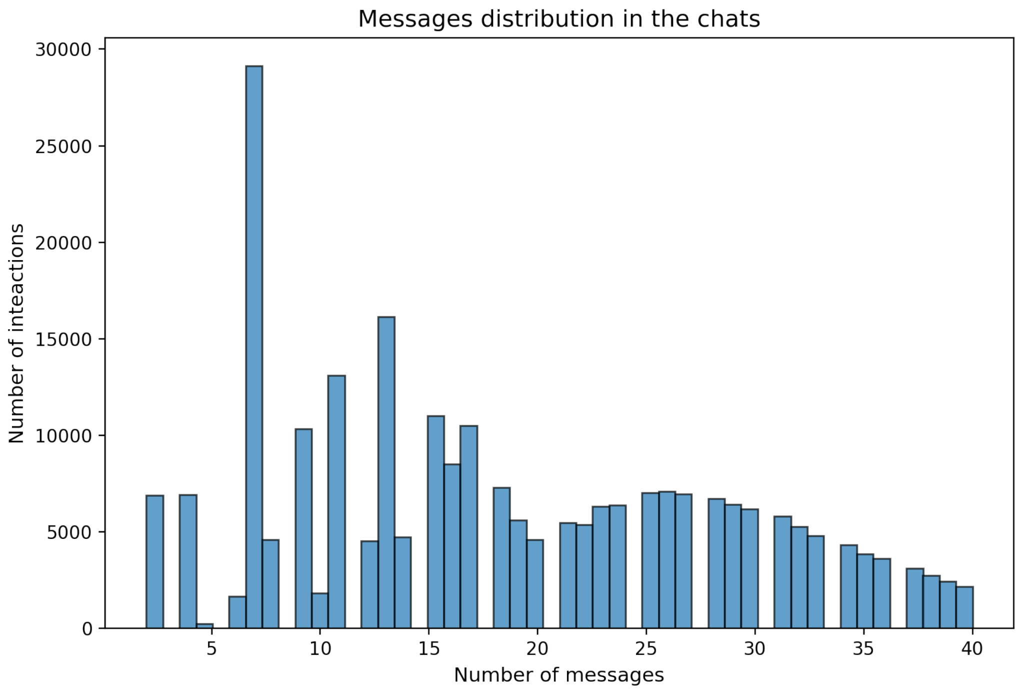


Figure 4.9: Number of messages in the chats.

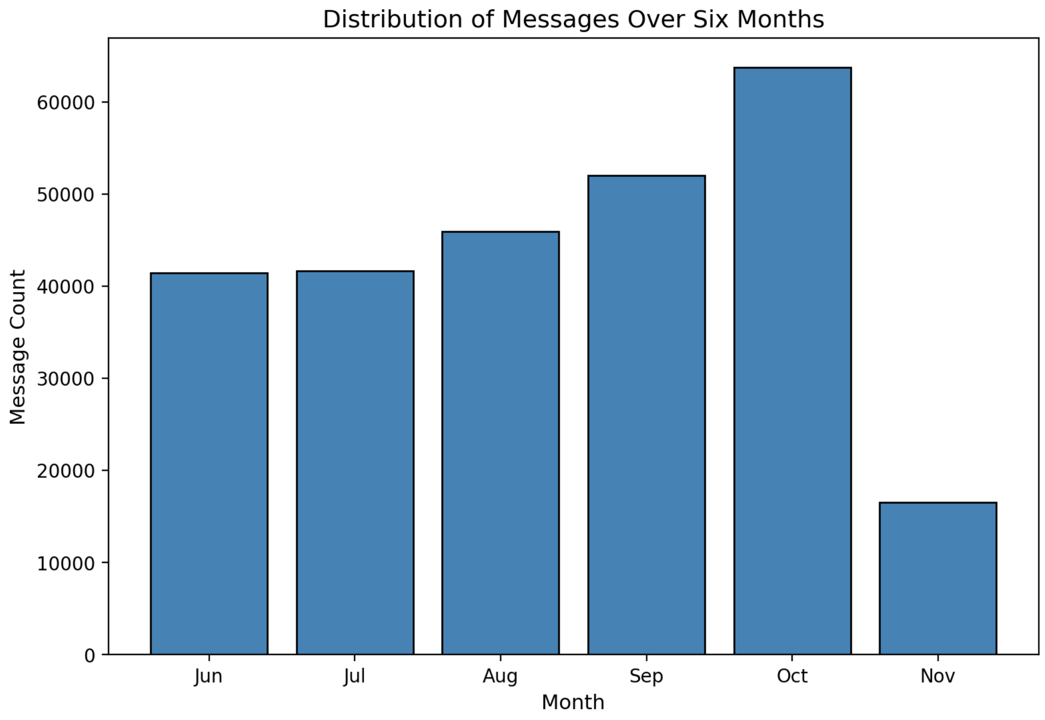


Figure 4.10: Messages distribution over months.

The analysis also included token counts for the messages, which prove valuable for cost evaluations. Indeed, the messages serve as inputs to the model within the prompt and costs for the Azure Open AI models are determined by the combined number of tokens in both input and output, as elaborated further in Section 5.1. Token counting was performed using the OpenAI *Tiktoken* library[1], which employs the same tokenizer utilized by GPT models in the experiment. The total token count is 37.514.722, with 141,25 average tokens per conversation.

From a temporal perspective, the data covers the period from June to the first half of November 2023, as illustrated in Figure 4.10, while the average number of calls per day is 1650.

Code written to compute statistics is reported in Appendix Section A.2.

[1]OpenAI Tiktoken library: `https://cookbook.openai.com/examples/how_to_count_tokens_with_tiktoken`.

4.4 Features Description

A crucial aspect of the solution design involved identifying specific features that the client aimed to derive for each conversation. As detailed in the preceding Section 4.1, the primary objective of the use case was to generate reusable features capable of serving as inputs for other analyses. To achieve this goal, a comprehensive series of meetings was arranged with the client company. To ensure a robust and well-defined framework that aligns both with business objectives and technical requirements, great importance was given to fostering collaboration between technical experts and professionals possessing domain knowledge specific to the legal gaming sector. The pivotal objective extended beyond the identification of features with business benefits and also involved the meticulous definition of the technical output format.

After several meetings and updates, nine target features were carefully selected and defined as follows:

1. ***Request Resolution*:** a measure of how effectively the customer's request is addressed, focusing on resolving the issues mentioned in the chat.

2. ***Customer Satisfaction*:** the customer's perceived satisfaction, deduced from their responses.

3. ***Customer Emotions*:** the emotions expressed by the customer in the conversation, coded according to Parrott's theory of emotions [Parrott, 2000]. Parrott's emotion model categorizes primary emotions into six classes: Love, Joy, Surprise, Anger, Sadness, and Fear. To provide a more comprehensive assessment of the feature, the emotion Love was excluded from the possible values, and a neutral emotion was added. Each conversation can be associated with one to three customer's emotions.

4. ***Customer Conversation Tone*:** the tone adopted by the customer in commu-

nicating during the conversation.

5. ***Customer Linguistic Level*:** describes the level of Italian language proficiency detected in the messages written by the customer.

6. ***Operator Emotions* :** defined as the prevailing emotional states of the operator, determined through the utilization of the customized Parrot emotion model, analogous to the method employed for assessing customer emotion.

7. ***Operator Conversation Tone*:** the tone adopted by the operator to communicate during the conversation.

8. ***Chat Summary*:** summary of the conversation with a focus on the topic.

9. ***Conversation Topic*:** one or two keywords extracted from the summary, identifying the main or the two main topics of the conversation.

Table 4.5 provides an overview of the potential values corresponding to each feature. For each feature, the value *NA* is designated for cases where they are not applicable or cannot be ascertained. For instance, features related to the operator should be marked as *NA* in conversations that conclude without the involvement of a human operator. The first seven features can be primarily associated with a classification task involving predefined classes, while the last two are freely generated by the model without any constraints. In more detail, the first two features, *Customer Request Resolution* and *Customer Satisfaction Perception* primarily revolve around customer satisfaction metrics. Features associated with tone and emotion are interrelated with the emotion detection task outlined in Section 3.1.1, while the *Customer Linguistic Level* feature pertains to the language competence of the customers. The last two features are essentially centered on the conversation's topic and are defined based on the tasks of summarization and keyword extraction, where GPT models exhibit high performance.

Feature	Possible Values
Request Resolution	Resolved, Unresolved, Not Resolvable, NA
Customer Satisfaction	Very Dissatisfied, Dissatisfied, Neutral, Satisfied, Very Satisfied, NA
Customer Emotions	J oy, Surprise, Anger, Sadness, Fear, Neutral, NA
Customer Conversation Tone	Formal, Informal, Emotional, Sarcastic, Ironic, NA
Customer Linguistic Level	Low, Medium, High, NA
Operator Emotions	J oy, Surprise, Anger, Sadness, Fear, Neutral, NA
Operator Conversation Tone	Low, Medium, High, NA
Chat Summary	*Text of max 180 characters,* *NA*
Conversation Topic	*Keyword extracted from the summary*

Table 4.5: Features and possible values.

4.5 Architecture Description

Regarding the architecture of the use case, the solution was designed consistent with the development of an Advanced Analytics system, as described in Section 2.2.

The designed architecture utilizes cloud services accessible through Microsoft Azure, with a focus on handling a substantial volume of data. This emphasis holds significance not only during the experimental phase but especially for subsequent applications. Figure 4.11 illustrates how all the Microsoft Azure services referenced in Section 2.3.2 are utilized to construct the proposed solution.

The data orchestration phase has been managed through Azure Data Factory, wherein a dedicated resource group containing all the services employed for the experiments is created. Among these resources, an *Azure Data Lake Gen2* serves as the storage system, housing data after the ingestion from the client's storage. Azure Databricks serves as a crucial analysis service and is employed for processing and managing the data, establishing a direct connection with the data lake storage. Leveraging Databricks notebooks written in Python, PySpark and Spark SQL, the data undergoes

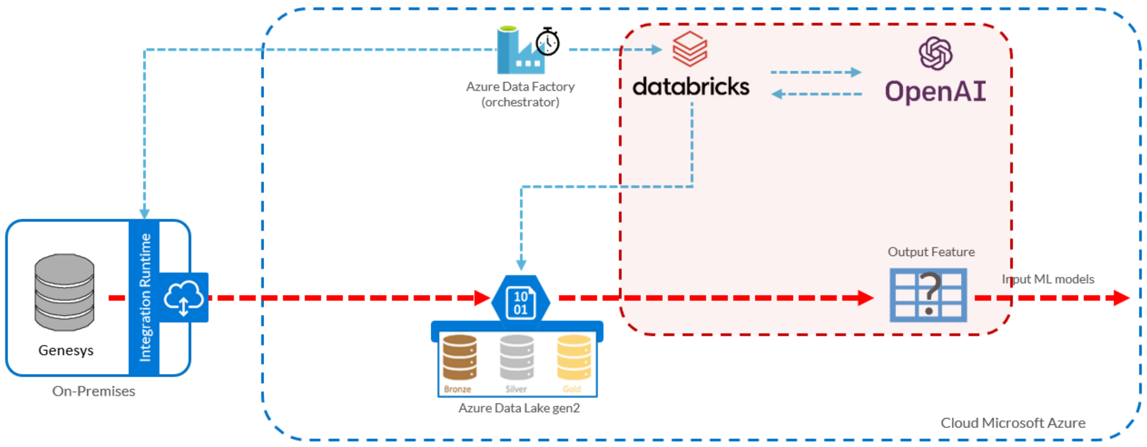


Figure 4.11: Use case Architecture.

exploration, cleaning, manipulation, and processing before being supplied as input to the model. Model interrogation and interaction are performed using Azure Databricks notebooks too, setting a communication with Azure Open AI. Each notebook establishes a connection with the storage, ensuring the storage of all intermediate and final outputs. The ultimate output is a dataset containing the nine features defined in Section 4.4, each valorized for every conversation. These features serve multiple purposes, enabling comprehensive business evaluations from various perspectives and facilitating further applications.

The company aims to integrate this architecture into its existing Machine Learning and advanced analytics framework. The idea is to incorporate the extracted features into a sophisticated analysis framework, using them as inputs for other Machine Learning and Artificial Intelligence models. This approach aims to maximize the information gained from the available data and establish connections between the results generated from all the company use cases. Future applications of the extracted features are described in more detail in Section 6.2.

Chapter 5

SOLUTION DEVELOPMENT

This chapter elucidates the core aspects of the use case, providing an in-depth exploration of the solution development across all phases. Section 5.1 delineates the model selection process from the options available through the Azure Open AI service, highlighting the parameters that led to the choice of GPT-4-Turbo for the experiments. Section 5.2 delves into the techniques, principles, and challenges encountered in crafting the prompts. The instruction written for the Large Language Model was undoubtedly a pivotal aspect of the experimentation process, evaluated through a framework detailed in Subsection 5.2.1. Subsection 5.2.2 outlines the final definition of the prompt used in the experimentation. Section 5.3 expounds the complete feature extraction process. The initial phase, detailed in Subsection 5.3.1, involves transforming the input, incorporating a sensitive data anonymization process and formatting conversations to make them suitable for model input. Subsection 5.3.2 describes the model interaction using the OpenAI Chat Completion API, while Subsection 5.3.3 explains how the model-generated output is reprocessed to obtain a dataset with the nine desired features for each conversation.

5.1 Model Selection

The heart of this experimental framework lies in the utilization of a Large Language Model for feature extraction. Given the pivotal role of the LLM, it was crucial to define the best way possible to include it in the advanced analytics framework, enhancing all its potential and flexibility to achieve the desired output smartly and effectively.

The selection of a suitable model from the options provided by Azure OpenAI for managing textual content was a critical decision. The potential candidate models for the experiment included OpenAI's GPT-3.5-Turbo-1106, GPT-4, and GPT-4-Turbo and all of them can be easily integrated into the project's architecture via Azure Open AI. It's important to note that at the time of the experiments, Azure Open AI GPT-4-Turbo was used in its preview state. However, since the full architecture is not yet close to production release, the experiments were still conducted using GPT-4-Turbo, anticipating a definitive release in the near future. To determine the optimal choice among the models, various parameters were taken into account.

The first evaluation metric was cost, which for Azure OpenAI models is based on the number of tokens. The cost, expressed in dollars, differs for prompts and completions and varies across different model versions. To provide an estimate of total and daily costs it has been employed the OpenAI *Tiktoken* library, already used for some of the statistics presented in Section 4.3. By applying the tokenization function with the *cl100k_base* encoding used by the candidate models, it is possible to compute an estimate of the number of tokens that the model would see both in input and output. This estimation was performed for 10.000 chats, considering a worst-case scenario by selecting the longest ones from the available sample. For each model interrogation, the input comprises the prompt, which is the one described in Section 5.2.2, and the chat to be analyzed. Consequently, for each chat, the fixed number of tokens in the prompt was added to the number of tokens in the messages, which instead changed in each interaction. For what completion estimates are concerned, a sample output

Model	Daily cost
GPT-3.5-Turbo-1106	5,33
GPT-4 (8k)	148,95
GPT-4 Turbo	55,32

Table 5.1: Approximate cost per day in dollars (rounded up).

was generated, formatted in JSON as described in Section 5.3.3, and multiplied by the number of chats.

Since the defined experiments were developed with the aim of creating an analysis system executable each day, the costs were estimated on a daily basis. Referring to the estimates calculated in Section 4.3, the average number of daily chats is assumed to be 1650 chats per day. Costs for the three models were estimated, based on the defined costs on the Azure Open AI website[1] and are presented in Table 5.1.

From the cost comparisons, it is evident that GPT-3.5-Turbo-1106 is undoubtedly the most economical model, while GPT-4 is the most expensive.

Other assessed parameters include the maximum prompt length, set at 16.000 tokens for GPT-3.5-Turbo-1106, 8.000 tokens for GPT-4, and 128.000 tokens for GPT-4-Turbo. The number of context tokens assumes particular relevance, allowing the possibility of crafting more detailed prompts or even embedding examples within them for *few-shot learning* as explained in Section 6.2.

Another crucial feature is the ability, limited exclusively to GPT-3.5-Turbo-1106 and GPT-4-Turbo, to define the desired output format by enforcing the generation of JSON content, as illustrated in Section 5.3.2. Since the desired output for each analysis is a JSON containing the features, this attribute proves particularly useful for managing automated analyses by minimizing the generation of exceptions that would later need post-processing.

[1]Azure OpenAI Service pricing: `https://azure.microsoft.com/en-us/pricing/details/cognitive-services/openai-service/`.

Regarding performance, a comparison was conducted on a sample of conversations between the performance of GPT-3.5-Turbo-1106 and GPT-4-Turbo, which appeared to be the two most appealing candidates. The evaluation was carried out using the same assessment framework described in Section 5.2.1 and determined better performance for GPT-4-Turbo, which has shown the ability to consistently define features more coherently, adhering more closely to the constraints of class definitions and better capturing the overall meaning of conversations.

In summary, considering all the previous factors, GPT-4-Turbo emerges as the selected model for the experiment. The model has a context of 128.000 tokens, a potential of 40.000 tokens per minute, a generation capacity of 4.096 tokens and it is trained on data until April 2023.

5.2 Prompt design and experiments

Since feature extraction is implemented using a generative LLM, formulating the instruction as a prompt has proven to be the most crucial and delicate phase of the experimentation. As explained in Section 3.4, generative LLMs require instructions in natural language to perform a task and the way this instruction is written can significantly impact the output generated by the model. In the current landscape, lots of best practices and techniques have emerged, offering valuable guidance for the art of crafting prompts. The evolving field of Natural Language Processing offers methodologies aimed at enhancing the clarity, specificity, and effectiveness of instructions provided to generative AI models, yielding optimal results comparable to human performances.

For this use case, the final goal of the prompt crafting process was to create a unified and clear instruction in Italian that maximized the model's understanding and performance across the diverse set of features, returning a semi-structured output in JSON format. The design of the prompt posed several challenges, which can be summarized into three primary focuses:

1. Determining the optimal number of instructions to write. Given the complexity of extracting nine features, a series of experiments were required to ascertain the most effective approach in crafting a single prompt for all features.

2. Refining the language and style used in instructions. Extensive testing has been conducted to identify the most clear and effective way to communicate with the model.

3. Selecting techniques for prompt crafting. A careful selection was made from state-of-the-art methods and best practices to incorporate into the process.

Addressing these challenges required a strategic and iterative approach to enhance the prompt's design, ensuring its effectiveness across multiple features. This process involved experimenting with different formulations, refining the language style and leveraging established techniques in the field. Indeed, the process for writing the instruction entailed continuous refinement of the prompt based on the obtained results, maintaining an ongoing discussion with the client company domain experts. In some cases this highly iterative process leads also to a revision of some feature definitions. For instance, emotions were initially defined for both primary and secondary extraction, which proved impractical and was revised based on tests that revealed consistent model pairings between certain emotions. Results evaluation was confirmed using a framework for the validation of experiments made by human annotators, as detailed in Subsection 5.2.1.

Number of prompts

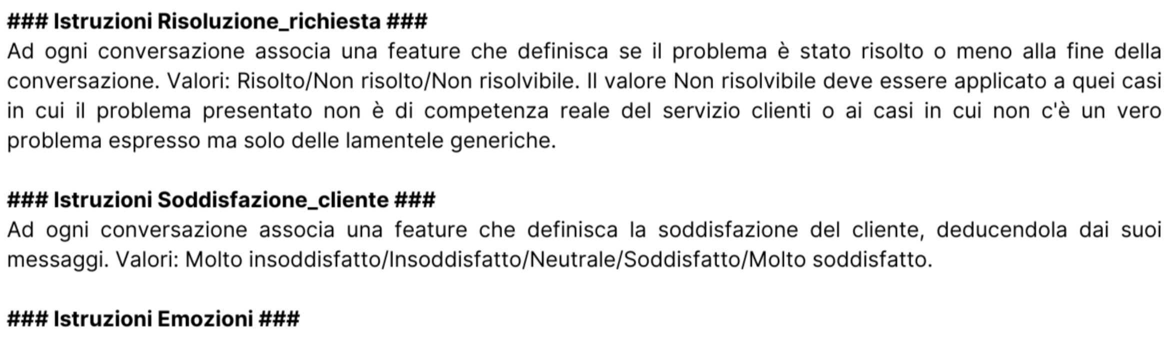
The first challenge addressed was determining how to write instructions for extracting the nine features. Even if each of the nine features requires the construction of an instruction for its definition, writing nine prompts would have incurred excessive costs. Therefore, the initial attempt was to group features that are semantically similar and write five prompts:

1. Prompt 1: *Request Resolution + Customer Satisfaction.*
2. Prompt 2: *Customer Emotions + Operator Emotions.*
3. Prompt 3: *Customer Conversation Tone + Operator Conversation Tone.*
4. Prompt 4: *Customer Linguistic Level.*
5. Prompt 5: *Chat Summary + Conversation Topic.*

Despite the good results obtained from the initial experiments, using five separate prompts proved to be still too costly. Indeed, employing five different prompts meant that the model received the same conversation as input five times with distinct instructions, leading to a quintuple rise in costs. Additionally, the post-processing operations would have been more intricate, requiring the merging of all results to obtain a unified dataset. For this reason, the experimentation shifted towards crafting a single instruction for extracting all nine features. The general approach taken for writing the prompt aimed to produce precise, highly structured, and concise text. Therefore, within the text, instructions for different features were differentiated using special characters as separators, an approach that is generally recommended to distinguish the instruction part from the input [Bsharat et al., 2023]. The idea is that a more structured division could help the model more clearly differentiate one feature from another. A single prompt for generating all nine features with the instruction separated with ### yielded excellent results and was thus adopted as the final solution. Figure 5.1 is extracted from the final prompt and shows how the separators are used for writing different instructions in the same text.

Prompt style

In the process of formulating the prompt, multiple sources were consulted, and several best practices were followed. The overarching goal was to create a well-organized, clear,



Istruzioni Risoluzione_richiesta
Ad ogni conversazione associa una feature che definisca se il problema è stato risolto o meno alla fine della conversazione. Valori: Risolto/Non risolto/Non risolvibile. Il valore Non risolvibile deve essere applicato a quei casi in cui il problema presentato non è di competenza reale del servizio clienti o ai casi in cui non c'è un vero problema espresso ma solo delle lamentele generiche.

Istruzioni Soddisfazione_cliente
Ad ogni conversazione associa una feature che definisca la soddisfazione del cliente, deducendola dai suoi messaggi. Valori: Molto insoddisfatto/Insoddisfatto/Neutrale/Soddisfatto/Molto soddisfatto.

Istruzioni Emozioni

Figure 5.1: Usage of separators for different instructions within the same prompt.

and concise instruction that empowers the model to execute the feature extraction task with a level of understanding similar to human capabilities. In particular, some of the principles outlined by Bsharat et al. [Bsharat et al., 2023] have been incorporated into the prompt design, including:

- Breaking down intricate tasks into a series of simpler instructions in a step-by-step manner.
- Favoring explicit positive requests ('do this') over negative constraints ('do not do this').
- Usage of delimiters to format prompts and clearly divide sections of the instruction.
- Usage of clear and concise language, avoiding useless information not regarding the task.
- Adding into the instruction some context information that could help the model understand the domain of the task.

To manage the generated output in a more defined way, explicit constraints related to the format and length of the desired features have been integrated into the instruction, following the recommendations of Ekin et al. [Ekin, 2023]. This also allows controlling the verbosity of the model to generate less or more information as needed.

The output format is also controlled through the *response format* parameter of the OpenAI API interacting with the model, as explained in more detail in Subsection 5.3.2.

The application of these best practices to the prompt is demonstrated in Subsection 5.2.2, where the final prompt used in the use case is showcased.

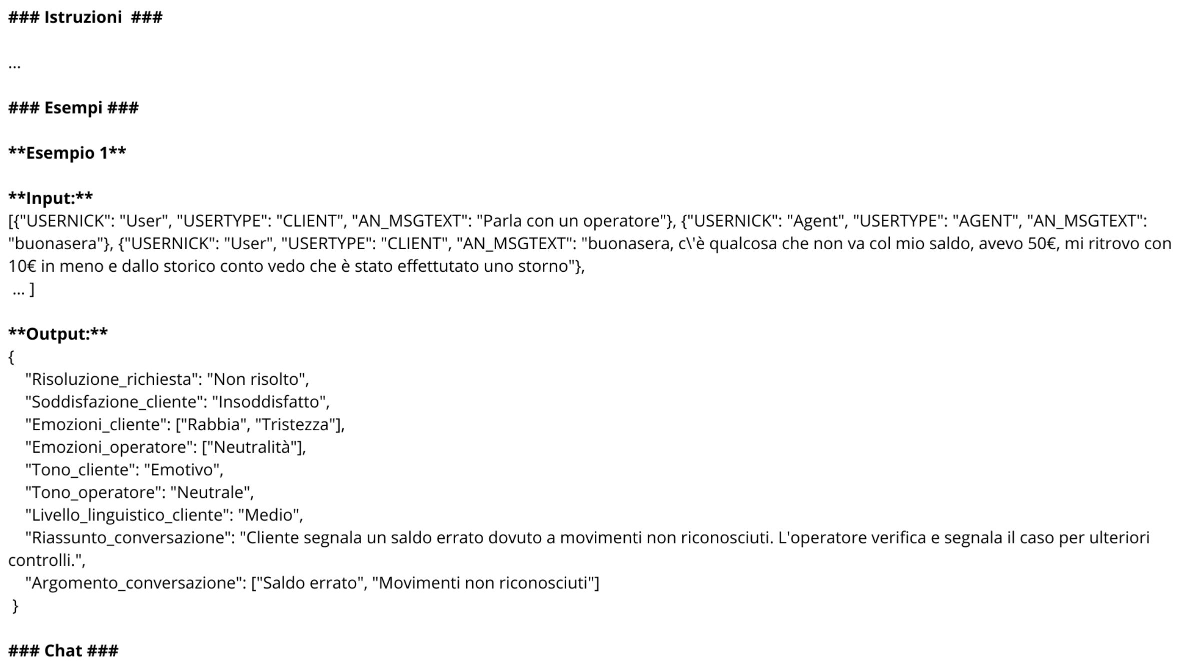
The integration of improvements into the prompt writing process was an iterative endeavor that involved consulting numerous sources and conducting a substantial number of trials. OpenAI's prompt engineering guide [Ope, 2024], in particular, proved to be a valuable source of inspiration for the numerous examples provided.

Prompting technique

Regarding the techniques used in the various prompt design experiments, the goal was to identify, among the numerous state-of-the-art techniques (illustrated in Subsection 3.4.1), those most suitable for the feature extraction task. This section of the study was particularly delicate, as the main focus of the experimentation was to create a prompt that would enable the generation of structured and precise results, leaving no room for other content outside the desired JSON output.

Some of the commonly used prompting techniques, such as *Chain-of-Thought* or *Self-Consistency*, do not align well with the specific task at hand and appear more suitable for mathematical or logical reasoning rather than textual analysis for feature extraction. Moreover, in the initial phase of the experimentation, the commissioning company expressed a preference for a cost-effective solution, so the selected final solution is a prompt written in zero-shot, illustrated in Section 5.2.2.

However, to explore potential future scenarios, a few-shot prompting approach was tested. This method involves providing correct examples from which the model can learn (see Subsection 3.4.1). To implement this technique, three examples were added at the end of the zero-shot defined prompt. These examples were integrated by adding



```
### Istruzioni ###

...

### Esempi ###

**Esempio 1**

**Input:**
[{"USERNICK": "User", "USERTYPE": "CLIENT", "AN_MSGTEXT": "Parla con un operatore"}, {"USERNICK": "Agent", "USERTYPE": "AGENT", "AN_MSGTEXT":
"buonasera"}, {"USERNICK": "User", "USERTYPE": "CLIENT", "AN_MSGTEXT": "buonasera, c\'è qualcosa che non va col mio saldo, avevo 50€, mi ritrovo con
10€ in meno e dallo storico conto vedo che è stato effettutato uno storno"},
 ... ]

**Output:**
{
   "Risoluzione_richiesta": "Non risolto",
   "Soddisfazione_cliente": "Insoddisfatto",
   "Emozioni_cliente": ["Rabbia", "Tristezza"],
   "Emozioni_operatore": ["Neutralità"],
   "Tono_cliente": "Emotivo",
   "Tono_operatore": "Neutrale",
   "Livello_linguistico_cliente": "Medio",
   "Riassunto_conversazione": "Cliente segnala un saldo errato dovuto a movimenti non riconosciuti. L'operatore verifica e segnala il caso per ulteriori
controlli.",
   "Argomento_conversazione": ["Saldo errato", "Movimenti non riconosciuti"]
 }

### Chat ###
```

Figure 5.2: Example of examples insertion in few shot prompt.

a block titled `###Examples###`, where each example, consisting of input and output, was preceded by a title with other separators such as `**Example 1**`. An illustration of how examples are provided in a few-shot prompt is presented in Figure 5.2. The input, presented in JSON format, mirrors the structure of a conversation as perceived by the model, while the output consists of a JSON dictionary with the desired features. Furthermore, to enhance clarity, both input and output segments have been distinctly delineated using separators, prefacing each section with `**Input**` and `**Output**`. Examples were chosen to be representative of a case with a clearly unsolved problem and negative customer emotions, a neutral case, and a case evidently resolved by assistance.

As explained in Section 6.2, this technique requires further exploration to be applied to this specific use case. From the few tests conducted, it seems that few-shot prompting has the potential to improve model performance, even with the associated increase in costs.

5.2.1 Experiments validation and evaluation

To assess the validity of the results obtained during the experimentation phase, an annotation schema was defined and carried out by human annotators. A sample of 40 conversations was extracted from the dataset and annotated by four human annotators defined as domain experts. Subsequently, the same 40 conversations were automatically annotated by GPT-4-Turbo using different prompts, followed by a qualitative evaluation of the results. The purpose of this evaluation system was both to validate the obtained results and to understand which aspects of each prompt can be improved, repeating the evaluation iteratively in case a better technique for writing the instruction was identified.

The conversations were selected to have a length coherent with the average length of the total sample and were manually analyzed to include examples of both very negative and very positive interactions, as well as neutral ones. The small number was chosen because the proposed evaluation process was set to be highly accurate, providing a qualitative assessment of each feature for each conversation by comparing it to human annotations. In fact, for each evaluation process, an initial exact match comparison was followed by a qualitative comparison to determine how much the automatic response differed from the human one. This dual control allowed distinguishing cases where the annotation could be considered completely incorrect from cases where it deviated slightly from the human one, to make improvements in the instruction for more severe cases.

This more structured type of evaluation was possible for the first seven features described in Section 4.4, excluding *Chat Summary* and *Conversation Topic*, for which a posteriori evaluation was proposed instead. While the first seven features were annotated and then the results from GPT were compared with manual annotations, these two generated features were directly assessed by domain experts to determine their suitability for the reference conversation.

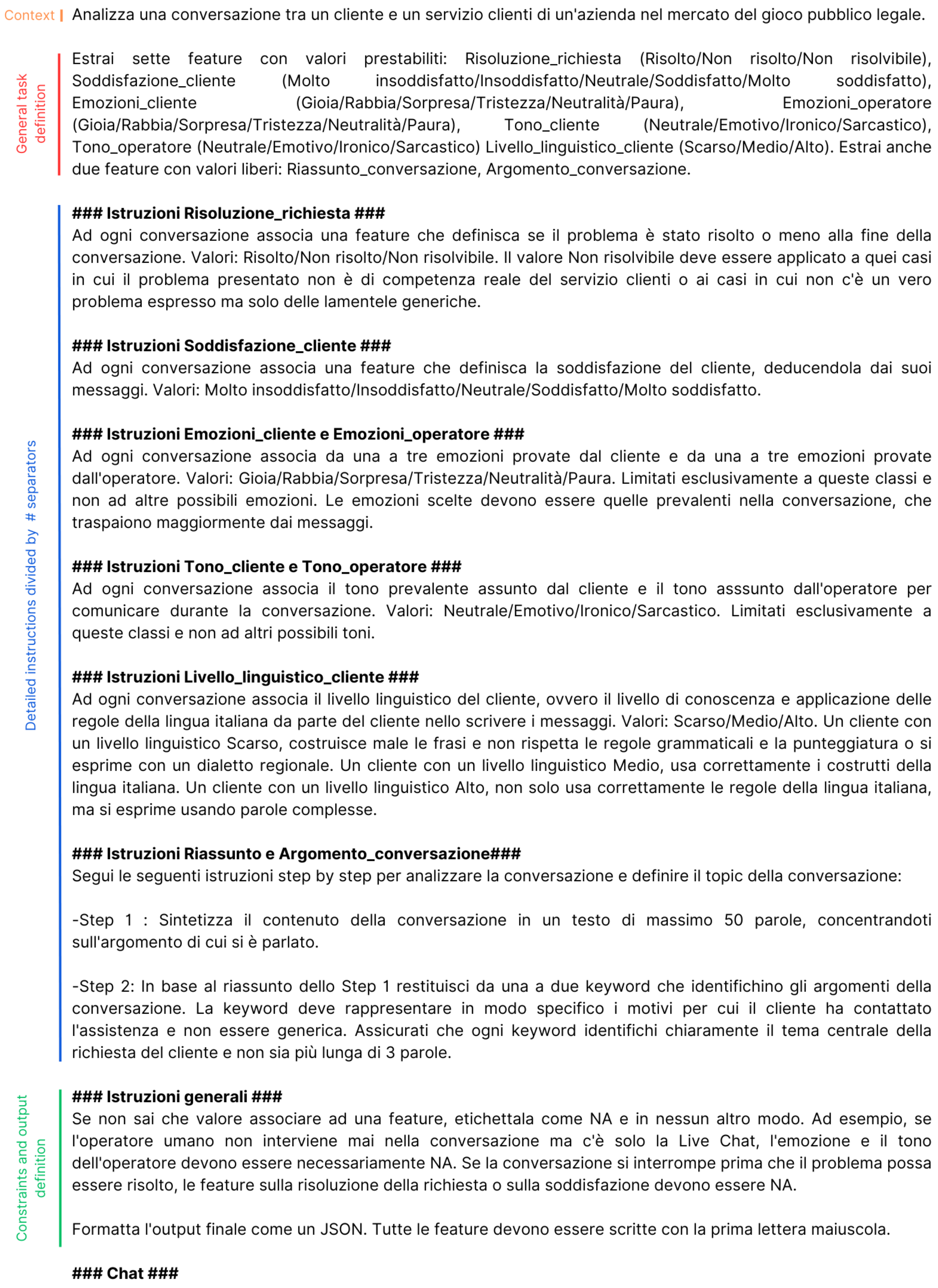
5.2.2 Final prompt considerations

After an extensive series of experiments, the final prompt used for the feature extraction task has been identified, as shown in Figure 5.3. The choice was made in collaboration with the domain experts from the company, who defined the instruction most suitable for the task.

Upon examining the instruction, is it possible to observe the application of principles outlined in the previous section to the written prompt. The initial part includes a contextual sentence to define the task's domain. Furthermore, all features have been pre-defined at the beginning of the instruction, with possible classes enclosed in parentheses. The general idea was to create a text as structured as possible, simplifying the task's execution and rendering it precise and well-defined. Instructions are separated using delimiters, also employed to introduce the text of each chat to be analyzed. For the features generated by the model, namely *Chat Summary* and *Conversation Topic*, a step-by-step approach was adopted. Indeed, after some experiments, it was observed that letting the model generate the descriptive keyword of the conversation from the summary led to significantly better results than seeking it within the conversation. For this reason, the instructions for these two features were placed within the same block, ensuring the generation of the keyword was consequential to the structure of the summary.

The evaluation framework proposed in Subsection 5.2.1 was also applied to the final instruction, yielding the results shown in Table 5.2.

Even if the written instruction already ensures satisfactory results, the prompt design remains subject to improvement within the context of an evolving experimentation process, as further illustrated in Section 6.2. Considerations about the output returned by experiment on a broader sample are contained in Section 6.1, focusing on the features that resulted to be more challenging for the model to be extracted. Figure 5.4 shows an example of completion returned by the prompt.


Context
General task definition
Detailed instructions divided by # separators
Constraints and output definition

Analizza una conversazione tra un cliente e un servizio clienti di un'azienda nel mercato del gioco pubblico legale.

Estrai sette feature con valori prestabiliti: Risoluzione_richiesta (Risolto/Non risolto/Non risolvibile), Soddisfazione_cliente (Molto insoddisfatto/Insoddisfatto/Neutrale/Soddisfatto/Molto soddisfatto), Emozioni_cliente (Gioia/Rabbia/Sorpresa/Tristezza/Neutralità/Paura), Emozioni_operatore (Gioia/Rabbia/Sorpresa/Tristezza/Neutralità/Paura), Tono_cliente (Neutrale/Emotivo/Ironico/Sarcastico), Tono_operatore (Neutrale/Emotivo/Ironico/Sarcastico) Livello_linguistico_cliente (Scarso/Medio/Alto). Estrai anche due feature con valori liberi: Riassunto_conversazione, Argomento_conversazione.

### Istruzioni Risoluzione_richiesta ###
Ad ogni conversazione associa una feature che definisca se il problema è stato risolto o meno alla fine della conversazione. Valori: Risolto/Non risolto/Non risolvibile. Il valore Non risolvibile deve essere applicato a quei casi in cui il problema presentato non è di competenza reale del servizio clienti o ai casi in cui non c'è un vero problema espresso ma solo delle lamentele generiche.

### Istruzioni Soddisfazione_cliente ###
Ad ogni conversazione associa una feature che definisca la soddisfazione del cliente, deducendola dai suoi messaggi. Valori: Molto insoddisfatto/Insoddisfatto/Neutrale/Soddisfatto/Molto soddisfatto.

### Istruzioni Emozioni_cliente e Emozioni_operatore ###
Ad ogni conversazione associa da una a tre emozioni provate dal cliente e da una a tre emozioni provate dall'operatore. Valori: Gioia/Rabbia/Sorpresa/Tristezza/Neutralità/Paura. Limitati esclusivamente a queste classi e non ad altre possibili emozioni. Le emozioni scelte devono essere quelle prevalenti nella conversazione, che traspaiono maggiormente dai messaggi.

### Istruzioni Tono_cliente e Tono_operatore ###
Ad ogni conversazione associa il tono prevalente assunto dal cliente e il tono asssunto dall'operatore per comunicare durante la conversazione. Valori: Neutrale/Emotivo/Ironico/Sarcastico. Limitati esclusivamente a queste classi e non ad altri possibili toni.

### Istruzioni Livello_linguistico_cliente ###
Ad ogni conversazione associa il livello linguistico del cliente, ovvero il livello di conoscenza e applicazione delle regole della lingua italiana da parte del cliente nello scrivere i messaggi. Valori: Scarso/Medio/Alto. Un cliente con un livello linguistico Scarso, costruisce male le frasi e non rispetta le regole grammaticali e la punteggiatura o si esprime con un dialetto regionale. Un cliente con un livello linguistico Medio, usa correttamente i costrutti della lingua italiana. Un cliente con un livello linguistico Alto, non solo usa correttamente le regole della lingua italiana, ma si esprime usando parole complesse.

### Istruzioni Riassunto e Argomento_conversazione###
Segui le seguenti istruzioni step by step per analizzare la conversazione e definire il topic della conversazione:

-Step 1 : Sintetizza il contenuto della conversazione in un testo di massimo 50 parole, concentrandoti sull'argomento di cui si è parlato.

-Step 2: In base al riassunto dello Step 1 restituisci da una a due keyword che identifichino gli argomenti della conversazione. La keyword deve rappresentare in modo specifico i motivi per cui il cliente ha contattato l'assistenza e non essere generica. Assicurati che ogni keyword identifichi chiaramente il tema centrale della richiesta del cliente e non sia più lunga di 3 parole.

### Istruzioni generali ###
Se non sai che valore associare ad una feature, etichettala come NA e in nessun altro modo. Ad esempio, se l'operatore umano non interviene mai nella conversazione ma c'è solo la Live Chat, l'emozione e il tono dell'operatore devono essere necessariamente NA. Se la conversazione si interrompe prima che il problema possa essere risolto, le feature sulla risoluzione della richiesta o sulla soddisfazione devono essere NA.

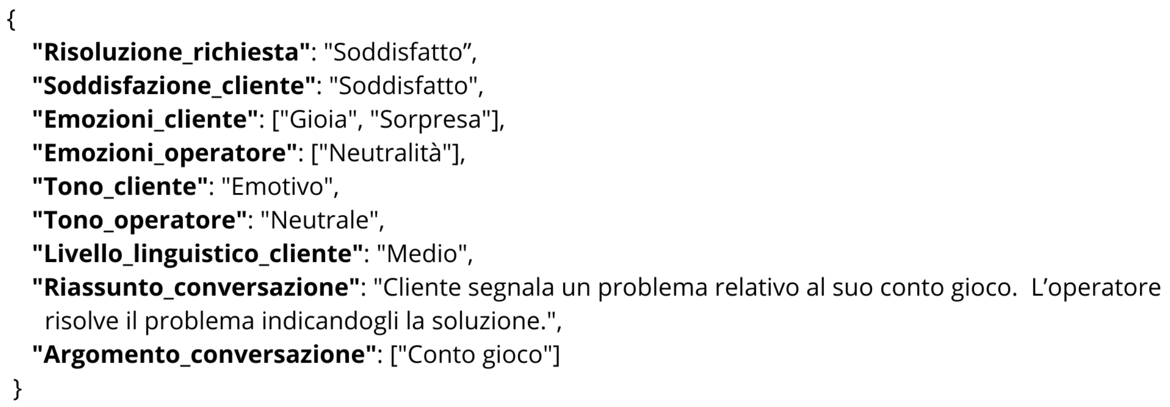
Formatta l'output finale come un JSON. Tutte le feature devono essere scritte con la prima lettera maiuscola.

### Chat ###

Figure 5.3: Final prompt for Features extraction.

Feature	Correct annotations
Risoluzione_richiesta	80% (32/40)
Soddisfazione_cliente	88% (35/40)
Emozioni_cliente	90% (36/40)
Tono_cliente	93% (37/40)
Livello_linguistico_cliente	90% (36/40)
Emozioni_operatore	95% (38/40)
Tono_operatore	95% (38/40)

Table 5.2: GPT-4-Turbo performance on a sample made of 40 chats.



```
{
  "Risoluzione_richiesta": "Soddisfatto”,
  "Soddisfazione_cliente": "Soddisfatto",
  "Emozioni_cliente": ["Gioia", "Sorpresa"],
  "Emozioni_operatore": ["Neutralità"],
  "Tono_cliente": "Emotivo",
  "Tono_operatore": "Neutrale",
  "Livello_linguistico_cliente": "Medio",
  "Riassunto_conversazione": "Cliente segnala un problema relativo al suo conto gioco.  L’operatore
    risolve il problema indicandogli la soluzione.",
  "Argomento_conversazione": ["Conto gioco"]
 }
```

Figure 5.4: Example of completion returned by GPT-4-Turbo with the designed prompt.

5.3 Features extraction workflow

The process of extracting the nine features from the chats, as defined in Section 4.4, forms the core of the advanced analytics architecture developed for this use case. The challenge was to create a fully automated process that could be replicated quickly and flexibly, even on a daily basis.

The feature extraction process comprises three main phases:

1. input pre-processing, detailed in Subsection 5.3.1;

2. model interaction, explained in Subsection 5.3.2;

3. output post-processing, illustrated in Subsection 5.3.3.

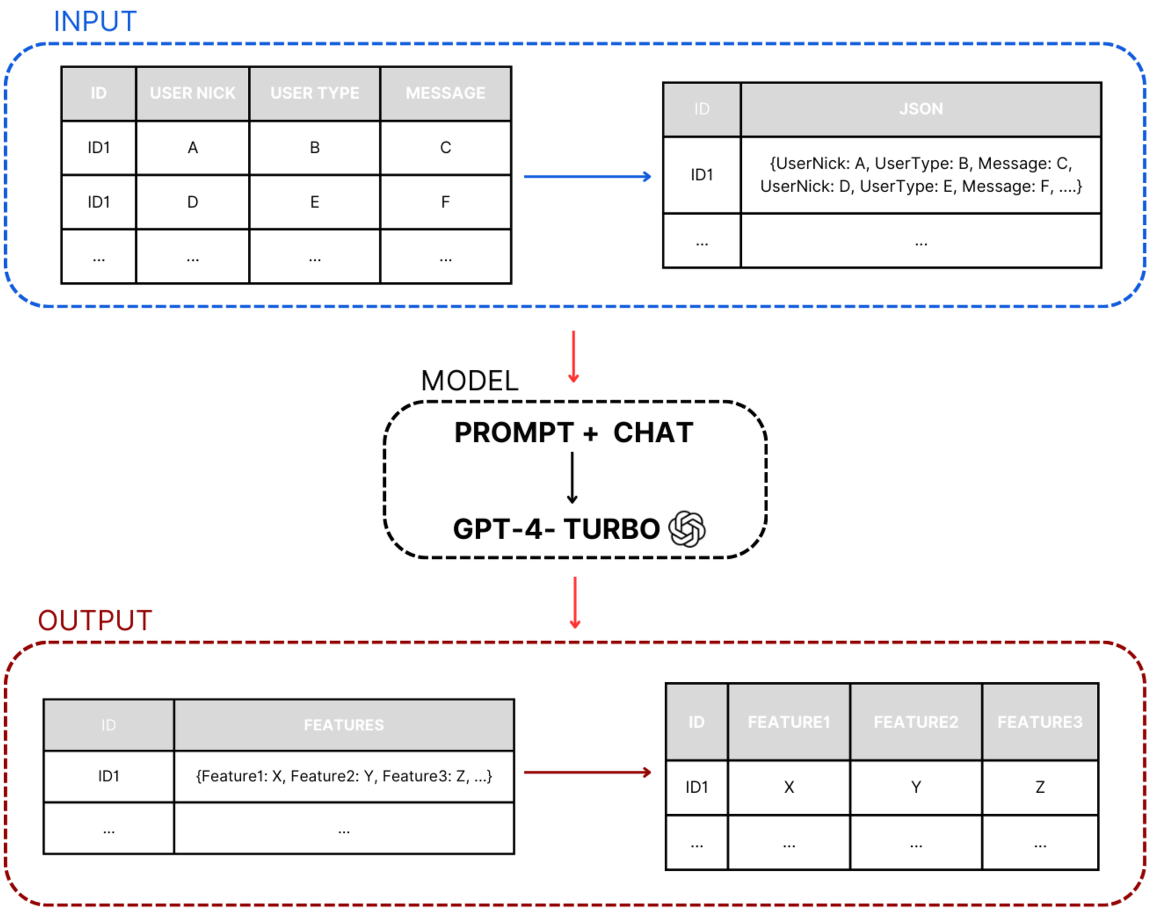
The integration of these operations is depicted in Figure 5.5, with a workflow starting from a structured dataset described in Section 4.2 and culminating in another dataset consisting of one conversation per row with the nine features extracted from the model. All operations were performed using a Databricks notebook scripted in Python, making particular use of the Python Pandas library for data manipulation[2].

5.3.1 Input preprocessing

Before being processed by the model the data undergoes several pre-processing operations, with the initial step involving anonymization.

Indeed, the employment of Language Models like GPT within a business framework raises significant concerns regarding data privacy, with a primary risk being the inadvertent disclosure of sensitive information related to customers. This confidential user data, commonly known as personally identifiable information (PII), is particularly likely to be present in datasets like customer support chats, as employed in this specific use case. In order to safeguard the data privacy of the company's customers, specific

[2]Pandas documentation: `https://pandas.pydata.org/`.



ID	USER NICK	USER TYPE	MESSAGE
ID1	A	B	C
ID1	D	E	F
...	...	...	...

ID	JSON
ID1	{UserNick: A, UserType: B, Message: C, UserNick: D, UserType: E, Message: F, ...}
...	...

ID	FEATURES
ID1	{Feature1: X, Feature2: Y, Feature3: Z, ...}
...	...

ID	FEATURE1	FEATURE2	FEATURE3
ID1	X	Y	Z
...	...	...	...

Figure 5.5: Feature extraction process.

measures are implemented to identify and anonymize PII within the chat. The following categories are defined as sensitive customer information within the chats and are subsequently anonymized:

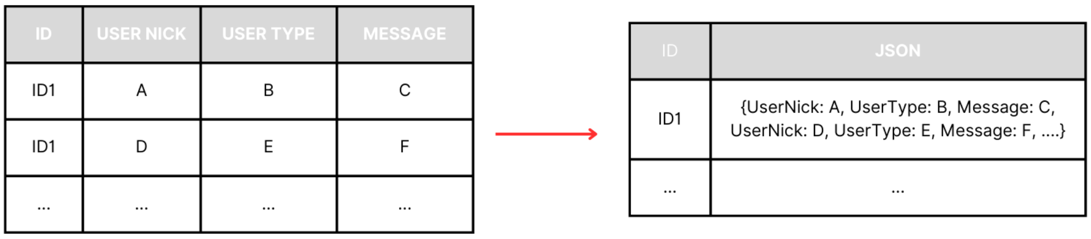
- Italian tax code;
- email addresses;
- credit card numbers;
- IBANs;
- links;
- phone numbers.

These sensitive data are identified using Regular Expressions (Regex), which are sequences of characters used to detect patterns in textual data. Whenever a pattern among the sensible data is detected, it is replaced with a generic string describing the class of that pattern. The code used to anonymize sensitive data is reported in A.3.

Once anonymized, the chat data are transformed to be in a suitable format for input into the model. The approach involves selecting only the necessary columns for analysis, grouping the rows by interaction ID, and transforming the data into a semi-structured format, such as JSON, to be able to input it into GPT in the prompt.

Converting the PySpark dataframe in a Pandas dataframe, these operations are obtained through a few lines of Python code, encoding data with a call to JSON API using `Json.dumps`[3]. Figure 5.6 shows an example of the preprocessing process from the anonymized dataset to the JSON format. While in the original data, each row contains a single message from the interaction, the new input dataset is composed of a single row for each interaction with the JSON field containing the whole conversation. For further details, pre-processing code is shown in A.4.1.

[3]JSON API for encoding and decoding: `https://docs.python.org/3/library/json.html`.



ID	USER NICK	USER TYPE	MESSAGE
ID1	A	B	C
ID1	D	E	F
...	...	...	...

ID	JSON
ID1	{UserNick: A, UserType: B, Message: C, UserNick: D, UserType: E, Message: F, ...}
...	...

Figure 5.6: Data preprocessing for model interaction.

5.3.2 Model interaction

The integration of the selected LLM, GPT-4-Turbo, within the architecture of advanced analytics for this use case is executed via the Azure Open AI cloud service, invoking the model using the OpenAI *Chat Completions* API[4].

This process involves sending requests to the OpenAI API, providing the necessary parameters such as model ID, maximum tokens, response format or temperature. The API then processes these inputs and returns an answer with the model-generated message under the specified configurations. The primary input of the API call is the *messages* parameter, an array of message objects. Each object within this array must specify a role, choosing between *system*, *user*, or *assistant* and include corresponding content. Although originally designed for multi-turn conversations, the chat format proves equally effective for single-turn tasks. The request body includes also other parameters, such as:

- *model*: ID of the model to use;
- max_tokens: the maximum number of tokens allowed in the generated completion;
- *response_format*: the desired format for the model's output. This feature is compatible only with GPT-4-1106-preview and GPT-3.5-turbo-1106. Opting for

[4]OpenAI Chat Completion documentation: `https://platform.openai.com/docs/api-reference/chat/create`.

`"type": "json_object"` activates JSON mode, ensuring that the model-generated message adheres to a valid JSON structure. It is crucial when using JSON mode to always instruct the model to produce JSON via some message in the conversation;

- *temperature*: a value between 0 and 1 which defines the randomness of the generated text. Higher temperature values return more random output, while lower temperature guarantees a more deterministic response;
- *top_p*: a value between 0 and 1 which stands as an alternative to temperature. This parameter defines which is the top_p probability mass of the tokens that the model has to consider. For example, setting it to 0.1 means only tokens representing the top 10% probability mass are taken into account. Generally, it is recommended to modify either this parameter or temperature, but not both.

The model yields a response object, containing a wealth of information whose primary is the message. This iterative exchange allows for dynamic and tailored interactions with the Language Model, enabling the extraction of relevant features from the input chats.

To incorporate it into the feature extraction workflow, the API Chat Completions call has been encapsulated within a function, defining both the optimal parameters for the model and the handling of potential errors. The function takes two messages as parameters: one is the system message, consisting of the prompt instruction detailed in Section 5.2.2, and the other is the user message, therefore each chat transformed in JSON as previously illustrated. Figure 5.7 illustrates how the prompt and the chat are input in the model. Other set parameters are the response format, defined as JSON, the maximum number of output tokens, set to 1000 and the temperature, set to 0 to guarantee a more deterministic output. The function includes the handling of generic exceptions, which is also necessary due to potential errors during experimentation

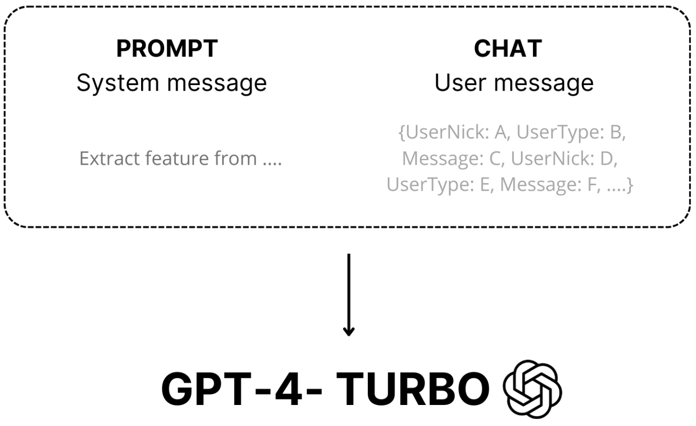
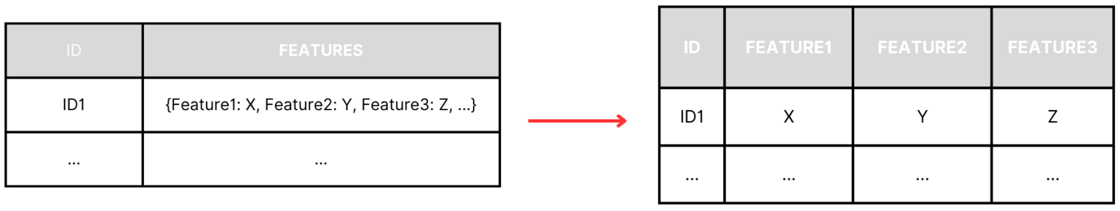

PROMPT
System message
Extract feature from ....
CHAT
User message
{UserNick: A, UserType: B, Message: C, UserNick: D, UserType: E, Message: F, ...}
GPT-4- TURBO

Figure 5.7: Model interaction.

related to sensitive content in the input that may violate Azure OpenAI policies[5]. Managing these cases is particularly intriguing since identifying them could pave the way for very useful applications of the analysis infrastructure. For instance, among the content filtered by Azure Open AI are materials associated with extreme violence and self-harm, which could be used by the company to be aware of users exhibiting such behaviors concerning their customer service. In fact, for future developments, it could be considered to implement specific triggers activated by contents of this nature to safeguard customers through personalized assistance services, as further detailed in Section 6.2. For these reasons, exception cases have been handled saving errors and details as responses.

The function for generating output is employed on JSON-formatted chats using the Pandas `apply`[6] method, also including a mechanism for detecting execution time using the tqdm Python library [7]. The code of the function for generating the output written for the experiments and the way it is applied to each interaction is illustrated in A.4.2.

[5]Azure Open AI Content filtering documentation: `https://learn.microsoft.com/en-us/azure/ai-services/openai/concepts/content-filter?tabs`.
[6]Pandas Apply documentation: `https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.apply.html`.
[7]tqdm documentation: `https://tqdm.github.io/`.



ID	FEATURES
ID1	{Feature1: X, Feature2: Y, Feature3: Z, ...}
...	...

ID	FEATURE1	FEATURE2	FEATURE3
ID1	X	Y	Z
...	...	...	...

Figure 5.8: Output manipulation.

5.3.3 Ouptut post-processing

GPT output is returned in JSON format, consisting of a dictionary with nine pairs of key values. Data is post-processed using the decoding `Json.loads` from JSON API[8]. The JSON API call is encapsulated within a function to handle potential values not in JSON format since, as demonstrated in Subsection 5.3.2, the function managing GPT output is designed to track possible errors returned by the API chat completions, which may not be encoded in JSON. For this reason, exception handling also involves adding a column to the dataframe indicating the success or failure of the decoding operation, allowing quick monitoring of interactions that may encounter issues. At this point, the JSON content returned by GPT as output is normalized using the Pandas `JSON_normalize` method[9], and the final output, as shown in Figure 5.8, is a dataframe containing the interaction ID, the success flag of the operation, and nine columns containing the extracted features. Post-processing code is illustrated in A.4.3.

[8]Refer to footnote 3 for JSON API for encoding and decoding documentation.

[9]JSON normalize Pandas method documentation: `https://pandas.pydata.org/docs/reference/api/pandas.json_normalize.html`.

Chapter 6

CONSIDERATIONS ON THE EXPERIMENTS

The current Chapter include comments related to the implementation of the analysis system and its potential improvements and future applications. Section 6.1 presents observations from the initial practical experiments using the GPT-4-Turbo-based feature extraction system. An overview of the general quality of the output is provided, addressing both content and format perspectives and highlighting major challenges. Section 6.2 suggests potential enhancements to the system from a technical standpoint and explores the possibilities of such an output and system in a broader business context.

6.1 Insights on the extracted features

The application of the feature extraction process on a larger sample of conversations has allowed for more detailed conclusions regarding the quality of the experiment. The feature extraction workflow has proven generally effective, supported by the ability to define a semi-structured output format like JSON as a parameter of the model.

Additionally, including a Success flag column in the final dataset makes it easy to manage exceptions during post-processing, filtering only the correct results. In any case, instances of failure were never found to be related to difficulties in processing the generated output unless it indicated an error generated by the Azure Open AI content filter (see Subsection 5.3.2), confirming the validity of the data manipulation process.

Interaction with a generative LLM poses significant challenges in the context of a classification with pre-defined categories. Indeed, for the first seven features, predefined classes are required to be generated and GPT-4-Turbo generally adheres to the existing classes without creating new ones. However, despite the clear and binding instructions written in the prompt, sometimes GPT generates different values for the features, not included in the possibility outlined in the prompt. The feature for which the model most frequently generates new classes is *Customer Emotions*, with the model often attempting to define these emotions more precisely than suggested. However, from the analyzed cases, it has been observed that such instances are rare, and the generated features are always consistent with the context of emotions and genuinely suitable for the associated conversation. In an experimental context like this use case, such cases are interesting as they may provide an opportunity to redefine the desired output, perhaps by extending the possible classes to include some of the cases more accurately recognized by the model. This behavior is more commonly observed concerning this feature likely because it is the most challenging to define, especially considering the domain of the texts. Indeed, online messages with customer care are often written in a casual style by customers and frequently involve technical problems and difficulties, not always revealing plainly the emotion proved by the writer.

Regarding the features to generate, specifically those related to the topic, the model demonstrates excellent performance. Since it's not possible to conduct a rigorous evaluation of these outputs, domain experts from the company have individually assessed some of the results, finding them satisfactory. The decision to let the model generate keywords rather than proposing a classification with predefined classes has proven to be very interesting and highlights the Language Model's ability to grasp the central aspects of the summary to define its main topics. Furthermore, the generated keywords exhibit a remarkable coherence: as expected, they are often repeated across different conversations and manage to be enough specific yet applicable to multiple cases simultaneously. It is noteworthy how the model consistently adheres to the maximum allowed length for emotion summaries and automatically generates a list of keywords related to the interaction topic, even in cases where it consists of only one element.

GPT-4-Turbo has demonstrated a solid understanding of the use of the *NA* value, intended to mark cases where a feature is not applicable and instances of conversations where all features are labeled as *NA* are quite rare.

In any case, the first seven features generated by the model are easily manageable in post-processing, as they have predefined classes to which they can be traced in case of minimal generation errors, which, however, did not occur in the conducted experiments. Features associated with summarization and keyword extraction pose a greater challenge in terms of control, requiring more extensive human validation in this initial phase.

The human validation must be extended to the entire feature extraction system in this early experimental phase, notwithstanding the positive outcomes already achieved.

6.2 Potential improvements and future applications

The feature extraction architecture described in the preceding sections was developed as part of an experimental phase with the client company, aiming to integrate the architecture into the advanced analytics infrastructure already implemented by the company.

Given the experimental nature of these activities, the obtained results are certainly subject to improvement and are intended to be an ongoing process, continuously refined and integrated over time. From a technical point of view, one potential enhancement for the future is the refinement of the prompt design, which could be incorporated into the architecture using a few-shot version. Few experiments have already been conducted in this sense, as illustrated in Section 5.2, by integrating labeled examples within the prompt to improve model performance. However, further refinement and evaluation on a larger sample are necessary for more precise conclusions. Another possible improvement is represented by a growing trend gaining popularity today, which involves enhancing the knowledge of Large Language Models by incorporating contextual information when providing input. This approach, known as Retrieval-Augmented Generation (RAG)[Lewis et al., 2020], has emerged as a promising solution that integrates knowledge from external databases. This technique allows Large Language Models to leverage additional data resources without retraining, contributing to more contextually rich and informed outputs.

Another path for enhancement involves personalized fine-tuning of the Large Language Model, which implies showing the model a broader sample of correctly annotated examples compared to those that can fit into the prompt. As suggested by Open AI[1], utilizing a fine-tuned model allows avoiding the need to over-provide examples in the prompt each time, thereby saving on costs and speeding up the process.

In general, extracting features from customer care-related material opens the doors to the development of numerous new internal use cases. Detecting features from online interactions could undoubtedly enhance the overall information assets of the company's analytics architecture for the benefit of all analysis operations. Creating new data can be useful, for example, to train other Machine Learning models related to the customer care domain, considering the implementation of tailored bonuses and campaigns based on the insights gathered. In addition to serving as input for new models, the analytical value of such a dataset is undoubtedly substantial and can help determine the overall

[1]Fine-tuning Open AI GPT models: `https://platform.openai.com/docs/guides/fine-tuning`.

trend of customer service concerning certain business operations or specific products and services.

Furthermore, the use of such innovative systems in a context like legal public gaming is linked to an ethical and benevolent use of AI. For instance, by leveraging the content filtering capabilities provided by Azure Open AI, as discussed in Subsection 5.3.2, it is possible to envision the real-time detection of potentially harmful or violent user behaviors. This may result in specific actions taken in such cases, such as utilizing a support mailing service specifically designed to address content related to suicidal tendencies. Of course, the development of such solutions still requires human supervision to determine the tool's accuracy in detecting these types of behaviors, to prevent the emergence of biases and discriminations. Nevertheless, it opens the doors to very interesting possibilities.

Chapter 7

CONCLUSIONS

The presented work provides an overview of the development of a novel advanced analytics system for extracting features from Italian texts obtained from customer service interactions using GPT-4-Turbo. The primary contribution of the study lies in integrating a generative Artificial Intelligence model as a tool for feature extraction from texts in a rigorous, structured, and precise manner. The focus was on determining the most effective way to write a prompt capable of delivering the desired output. In this regard, the experimentation required significant prompt engineering, involving an iterative process of instruction writing that was continuously refined and adapted to be as structured as possible. Various techniques and state-of-the-art best practices were tested, drawing upon thorough research and documentation related to prompt design. It was particularly interesting to observe that even a zero-shot prompt, written without the administration of any example but with precise and concise language, yielded highly satisfactory results, leaving room for the implementation of multiple improvements.

The conducted study has demonstrated how generative Large Language Models can now be integrated into a business context, making the execution of tasks that would have previously required much more effort and resources quick and straightforward. In fact, without the need for any retraining or significant endeavors for input and output manipulation, it is possible to integrate a tool into an analytics architecture that only requires the definition of a prompt to perform a wide range of tasks related to textual data. The architecture and training of this class of models allow for a highly efficient response to the scarcity of training data in various sectors by developing analyses based on the model's prior knowledge and the precise definition of the task to be performed. The use of such technologies enables the comprehensive exploitation of an organization's data, paving the way for numerous new use cases that leverage the outputs of previous use cases.

One of the most challenging aspects of these experiments is the generative nature of the models, which is an extremely valuable feature but can lead to a deviation from the intended value suggested in the prompt. This tendency can be interesting to expand the set of notable classes in an experimental phase but also problematic in a rigorous well-structured classification context. The experiments conducted suggest that this challenge can be addressed with prompt engineering. However, it is important to note that even after suitably crafting a prompt, the generation of undesired content cannot be completely avoided. Additionally, the pivotal role of the prompt is a double-edged sword: while it makes the base architecture adaptable to any task by simply changing the instruction, the way in which the task is described has a profound impact on the model's performance, making crucial to craft the prompt as accurately as possible.

Considering the experimental context of this use case, the obtained results can certainly be enhanced through the application of suitable techniques. Indeed, to improve the outcomes a spectrum of methodologies can be employed, encompassing not only the refinement of the model through fine-tuning but also the augmentation of its contextual information, incorporating strategies that extend beyond mere model adjustments for implementing a more comprehensive and effective task execution. Moreover, the dynamic and ever-evolving nature of this field leaves the door open to numerous new possibilities, as it is anticipated that research will undergo significant advancements in the near future.

In conclusion, Large Language Models are undoubtedly a valuable resource not only in a research context but also in a business environment, streamlining and improving various analytical processes. However, their usage must be judicious and always guided by expert opinion, especially when handling data related to individuals. This case study exhibits how generative Artificial Intelligence can be a substantial asset, working in conjunction with qualified human specialists and contributing to the development of cutting-edge, fast, and flexible analytical systems.

Acknowledgements

I sincerely thank Advant for providing me with the opportunity to enhance my skills through this thesis project and for supporting me throughout every stage of the work. In particular, I express my gratitude to Nicola Forestieri and Viktoryia Kotava, with whom I had the opportunity to work directly, meticulously defining the phases of this experimentation.

I also extend heartfelt thanks to my advisor, Professor Claudio Gallicchio, for guiding me during the thesis writing process, providing valuable insights into potential refinements for the project and broadening my knowledge in the field of Natural Language Processing.

I genuinely express my gratitude to all those who have contributed to my education, allowing me to expand my knowledge also beyond the traditional academic context. In this regard, special thanks go to Professor Lucia Passaro and Professor Paolo Ferragina for allowing me to delve into the world of research.

Appendix A

Experiments Code

A.1 Preliminary Data Preprocessing

As described in Section 4.2, the preprocessing of chats involved the removal of errors from the original sample and the imputation of missing values and has been implemented by exploring Apache Spark's SQL functionality. To begin, a temporary table has been created for SQL operations and all rows containing either complete null values or a single column set have been selected and stored in a separate dataset. This dataset containing errors has been subtracted from the original data, obtaining a corrected dataset. Subsequently, the dataset containing only the corrected rows has been transformed exploiting a new temporary table for SQL operations. The transformation involved handling missing values, casting data types and using window functions to fill in certain columns with the first non-null values within specific partitions. The resulting dataframe has then been ordered based on interaction ID and event ID, in order to make chats readable. The final output is a well-structured dataframe that has been cleansed of errors, filled with appropriate values and organized for improved coherence in analyzing chat.

```
#Reading a CSV file into a Spark DataFrame
```

```
df = spark.read.csv(file_path, header=True, inferSchema=True,
    sep=";")

#Creating a temporary table named "chat" for SQL operations
df.createOrReplaceTempView("chat")

#Filtering out rows with null values in specific columns
error_df = spark.sql("""
    SELECT *
    FROM chat
    WHERE INTERACTION_ID IS NULL OR USER_ID IS NULL OR TRY_CAST(
    DATE_TIME AS DATE) IS NULL
""")

#Subtracting the error rows from the original DataFrame to get
    correct data
dfcorrect = df.subtract(error_df)
dfcorrect.createOrReplaceTempView("corrects")

#Filling missing values in the dataset and creating a new
    DataFrame
df_filled = spark.sql("""
    SELECT
        DATETIME AS DATE_TIME,
        CAST(DATE_TIME AS DATE) AS DATE,
        SUBSTRING(DATE_TIME, 12, 8) AS TIME,
        INTERACTION_ID,
        USER_ID,
        CAST(TIMESHIFT AS INT),
        VISIBILITY,
        CAST(EVENT_ID AS INT) ,
        FIRST(PERSON_ID) OVER (PARTITION BY USER_ID ORDER BY
    CAST(EVENT_ID AS INT)) AS PERSON_ID,
        FIRST(USERNICK) OVER (PARTITION BY USER_ID ORDER BY CAST
    (EVENT_ID AS INT)) AS USERNICK,
        FIRST(USERTYPE) OVER (PARTITION BY USER_ID ORDER BY CAST
    (EVENT_ID AS INT)) AS USERTYPE,
        FIRST(PROTOCOLTYPE) OVER (PARTITION BY USER_ID ORDER BY
    CAST(EVENT_ID AS INT)) AS PROTOCOLTYPE,
        TZOFFSET,
        MSGTEXT,
        ASKERID,
        REASON
    FROM corrects
    ORDER BY INTERACTION_ID, EVENT_ID
""")
```

A.2 Statistics

Code for the statistics described in Section 4.3 has been written in PySpark, importing relevant functions from the PySpark library and then reading data from a CSV file into a Spark dataframe. Afterward, it has been created a temporary table to facilitate SQL operations on the dataframe.

```
from pyspark.sql.functions import count, countDistinct, avg, max
    as spark_max, min as spark_min, round, sum

#Reading a CSV file into a Spark DataFrame
df = spark.read.csv(file_path, header=True, inferSchema=True,
   sep=",")

```

```
#Creating a temporary table named chat for SQL operations
df.createOrReplaceTempView("chat")
```

A.2.1 Conversation statistics

Numerous statistical insights encompassing counts, averages, and durations have been computed to characterize different aspects of the conversation data.

```
#Number of distinct conversations
count_conv = df.select(countDistinct("INTERACTION_ID")).first()
    [0]

#Average number of distinct conversations per day
count_convD = df.groupBy("DATE").agg(countDistinct("
    INTERACTION_ID").alias("count")) \
     .agg(round(avg("count"), 2).alias("avg")).first()[0]

#Total null messages
count_null = df.filter(df["MSGTEXT"].isNull()).select(count("*")
    .alias("null_messages")).first()[0]

#Null messages which are not connected with the exit from the
    conversation
count_realnull = df.filter((df["MSGTEXT"].isNull()) & (df["
    REASON"].isNotNull())) \
.select(count("*")).first()[0]

#Number of conversations without human agents
chats_without_agents = "SELECT COUNT(DISTINCT INTERACTION_ID)
    FROM chat WHERE INTERACTION_ID NOT IN (SELECT INTERACTION_ID
    FROM chat WHERE USERTYPE='AGENT' AND USERNICK != 'Live Chat'
    GROUP BY INTERACTION_ID)"
n_chat_without_ag = spark.sql(chats_without_agents)
count_chat_without_ag = n_chat_without_ag.collect()[0][0]

#Average conversation duration in minutes
count_duration = df.groupBy("INTERACTION_ID").agg(
    ((spark_max("TIMESHIFT") - spark_min("TIMESHIFT")) / 60.0).
    alias("CHAT_DURATION")
).agg(round(avg("CHAT_DURATION"), 2).alias("average_duration")).
    first()[0]

#Longest conversation duration in minutes
max_duration = df.groupBy("INTERACTION_ID").agg(
    ((spark_max("TIMESHIFT") - spark_min("TIMESHIFT")) / 60.0).
    alias("CHAT_DURATION")
).agg(round(spark_max("CHAT_DURATION"), 2).alias("
    longest_duration")).first()[0]

#Shortest conversation duration in minutes
min_duration = df.groupBy("INTERACTION_ID").agg(
    ((spark_max("TIMESHIFT") - spark_min("TIMESHIFT")) / 60.0).
    alias("CHAT_DURATION")
).agg(round(spark_min("CHAT_DURATION"), 2).alias("
    shortest_duration")).first()[0]

#Average number of messages per conversation
count_mess = df.groupBy("INTERACTION_ID").agg(count("*").alias("
    CHAT_MESS")) \
.agg(round(spark_avg("CHAT_MESS"), 2).alias("average_messages"))
    .first()[0]

#Max number of messages per conversation
```

```
count_max = df.groupBy("INTERACTION_ID").agg(count("*").alias("CHAT_MESS")) \
    .agg(round(spark_max("CHAT_MESS"), 2).alias("max_messages")).first()[0]

#Min number of messages per conversation
count_min = df.groupBy("INTERACTION_ID").agg(count("*").alias("CHAT_MESS")) \
    .agg(round(spark_min("CHAT_MESS"), 2).alias("min_messages")).first()[0]
```

A.2.2 User statistics

Other statistics have been computed on the distribution of messages among different user types, identifying returning customers and calculating the count of distinct clients in the data.

```
#Number of human messages
count_human = df.filter((df["USERTYPE"] == "CLIENT") | (df["USERTYPE"] == "AGENT") & (df["USERNICK"] != "Live Chat")) \
    .select(count("*")).first()[0]

#Number of bot and system messages
count_sys = df.filter((df["USERTYPE"] == "EXTERNAL") | (df["USERNICK"] == "Live Chat")) \
    .select(count("*")).first()[0]

#Number of messages for different user types
user_types_mess = df.groupBy("USERTYPE").count()

#Returning customers
```

```python
ret_count = (df.filter((df["USERTYPE"] == "CLIENT")) \
    .groupBy("USERNICK").agg(countDistinct("INTERACTION_ID").
   alias("NumberOfConversations")) \
    .filter("NumberOfConversations > 1")).count()

#Distinct customers
count_client_dist = df.filter((df["USERTYPE"] == "CLIENT")) \
    .select(countDistinct("USERNICK")).first()[0]
```

A.2.3 Tokenization

Since the analysis included tokens count, the OpenAI *Tiktoken* library has been integrated to compute the number of tokens in text messages. The code defines a user-defined function named `num_tokens_from_string`, which takes a string as input and uses the library to count the number of tokens in that string. The function returns an integer representing the token count and has been used to create a new dataset by adding a new column named *token_ count* to the original dataframe, populated with the token counts calculated by applying the previously defined function to the *MSGTEXT* column. Using the new dataframe, they have been calculated the average number of tokens per conversation and the total number of tokens in the dataset.

```python
import tiktoken
import openai
from pyspark.sql.functions import udf
from pyspark.sql.types import IntegerType

#Function to count numer of tokens leveraging Open AI tiktoken
   library
@udf(IntegerType())
def num_tokens_from_string(string: str) -> int:
    if string is None:
        return 0
    else:
        encoding = tiktoken.get_encoding('cl100k_base')
        num_tokens = len(encoding.encode(string))
        return num_tokens

#New DataFrame with a column with tokens numbers for each message
df_token_counts = df.withColumn("token_count", num_tokens_from_string(df['MSGTEXT']))

#Average number of tokens per conversation
count_avgtok = df_token_counts.groupBy("INTERACTION_ID").agg(sum("token_count").alias("count")) \
    .agg(round(spark_avg("count"), 2).alias("average_tokens")).first()[0]

#Total number of tokens
count_tokentot = df_token_counts.agg(sum("token_count").alias("total_tokens")).first()[0]
```

A.3 Data Anonimization

Using a PySpark dataframe, sensitive information in messages has been identified and anonymized to protect user privacy, as detailed in Subsection 5.3.1. Sensitive data patterns have been defined using regular expressions for different types of sensitive information. Two columns have been added to the dataset: a new column *AN_MSGTEXT*, created by copying the original *MSGTEXT* column, and a *Pattern* column to identify which sensitive pattern has been found in each message, using the `when` and `rlike` functions and marking the pattern or *False* if none is found. Then it has been defined a list of tuples, each containing a sensitive data pattern and its replacement and, iterating through this list, occurrences of sensitive data patterns have been replaced with their corresponding replacements in the *AN_MSGTEXT* column.

```python
from pyspark.sql.functions import col, when, regexp_replace

#Define the sensitive data patterns using Regex
CODICE_FISCALE = r'\b(?:[A-Za-z][AEIOUaeiou][AEIOUXaeioux]|[AEIOUaeiou]X{2}|[B-DF-HJ-NP-TV-Zb-df-hj-np-tv-z]{2}[A-Za-z]){2}(?:[\dLMNP-Vlmnp-v]{2}(?:[A-EHLMPR-Ta-ehlmpr-t](?:[04LQ][1-9MNP-Vlmnp-v]|[15MR][\dLMNP-Vlmnp-v]|[26NS][0-8LMNP-Ulmnp-u])|[DHPSdhps][37PTpt][0L]|[ACELMRTacelmr-t][37PTpt][01LM]|[AC-EHLMPR-Tac-ehlmpr-t][26NSns][9Vv])|(?:[02468LNQSUlnqsu][048LQU048lqu]|[13579MPRTV13579mprtv][26NS26ns])B[26NS26ns][9Vv])(?:[A-MZa-mz][1-9MNP-Vlmnp-v][\dLMNP-Vlmnp-v]{2}|[A-Ma-m][0L](?:[1-9MNP-Vlmnp-v][\dLMNP-Vlmnp-v]|[0L][1-9MNP-Vlmnp-v]))[A-Za-z]\b'
EMAIL = r'\b[\w\-\.]+@([\w-]+\.)+[\w-]{2,}\b'
IBAN = r'\b[A-Za-z]{2}[0-9]{2}(?:[ ]?[0-9]{4}){4}(?!(?:[ ]?[0-9]){3})(?:[ ]?[0-9]{1,2})?\b'
VISA = r'\b4[0-9]{12}(?:[0-9]{3})?\b'
MASTERCARD = r'\b(5[1-5][0-9]{14}|2(22[1-9][0-9]{12}|2[3-9][0-9]{13}|' \
             r'[3-6][0-9]{14}|7[0-1][0-9]{13}|720[0-9]{12}))\b'
AMEX = r'\b3[47][0-9]{13}\b'
MAESTRO = r'\b(5018|5020|5038|6304|6759|6761|6763)[0-9]{8,15}\b'
VISA_MASTERCARD = r'\b(?:4[0-9]{12}(?:[0-9]{3})?|5[1-5][0-9]{14})\b'
PHONE_NUMBER = r'\b[\+]?[(]?[0-9]{3}[)]?[-\s\.]?[0-9]{3}[-\s\.]?[0-9]{4,6}\b'
```

```
LINK = r'(http|ftp|https):\/\/([\w_-]+(?:(?:\.[\w_-]+)+))([\w.,@
    ?^=%&:\/~+#-]*[\w@?^=%&\/~+#-])'

#Assuming df is a Spark DataFrame. Replace sensitive content
    found in the column MSGTEXT with a generic string a write them
     in a new column called AN_MSGTEXT
df = df.withColumn("AN_MSGTEXT", col("MSGTEXT"))

#Add a Pattern column which defines which pattern has been found
df = df.withColumn(
     "Pattern",
     when(col("MSGTEXT").rlike(CODICE_FISCALE), "Codice Fiscale")
     .when(col("MSGTEXT").rlike(EMAIL), "Email")
     .when(col("MSGTEXT").rlike(IBAN), "IBAN")
     .when(col("MSGTEXT").rlike(VISA) | col("MSGTEXT").rlike(
    MASTERCARD) | col("MSGTEXT").rlike(AMEX) | col("MSGTEXT").
    rlike(MAESTRO) | col("MSGTEXT").rlike(VISA_MASTERCARD), "
    Numero carta di credito")
     .when(col("MSGTEXT").rlike(PHONE_NUMBER), "Numero di
    telefono")
     .when(col("MSGTEXT").rlike(LINK), "Link")
     .otherwise('False')
)

#Define a list containing tuples with the pattern name and the
    replacement
patterns = [
     (CODICE_FISCALE, "Codice Fiscale"),
     (EMAIL, "Email"),
     (IBAN, "IBAN"),
     (VISA, "Numero carta di credito"),
     (MASTERCARD, "Numero carta di credito"),
    (AMEX, "Numero carta di credito"),
    (MAESTRO, "Numero carta di credito"),
    (VISA_MASTERCARD, "Numero carta di credito"),
    (PHONE_NUMBER, "Numero di telefono"),
    (LINK, "Link")
]

#Iterate through sensitive data patterns and replace them with
    generic strings
for pattern, replacement in patterns:
    df = df.withColumn("AN_MSGTEXT", regexp_replace(col("
    AN_MSGTEXT"), pattern, replacement))
```

A.4 Feature extraction workflow

For all its phases, the features extraction process described in Section 5.3 has been implemented using Python code and exploring different libraries, such as Pandas.

A.4.1 Data pre-processing for model interaction

Conversations have been preprocessed to be suitable for being used as input for GPT, in order to transform a Spark dataframe into a more structured and JSON-compatible format, as detailed in Subsection 5.3.1. First of all, has been defined the `convert_to_json` function, which takes a list and uses the `json.dumps` function to convert it into a JSON-formatted string. Then, the Spark dataframe has been converted into a Pandas dataframe and it has been defined a list containing the column that will be considered for analysis. The Pandas df has then been grouped by the interaction ID and the selected columns have been transformed into a list of dictionaries for each group using the apply method. This results in a dataframe named with a *chats* column containing lists of dictionaries. Finally, the `convert_to_json` function has been applied to the *chats* column, creating a new column named *json_string* that contains JSON-formatted strings representing the grouped and transformed data.

```python
import json
import pandas as pd

#Function to convert a list to a JSON string. The ensure_ascii=
    False parameter ensures that Unicode characters are not
    escaped and are represented directly in the output
def convert_to_json(my_list):
     return json.dumps(my_list, ensure_ascii=False)

#Assuming df is a Spark DataFrame, convert it to a Pandas
    DataFrame
df = df.toPandas()

#Define a list of selected columns for analysis
selected_columns = ['INTERACTION_ID', 'USERNICK', 'USERTYPE', '
    AN_MSGTEXT']

#Group the DataFrame by INTERACTION_ID and transform selected
    columns into a list of dictionaries
grouped_chats = df.groupby('INTERACTION_ID')[selected_columns
    [1:]].apply(lambda x: x.to_dict('records')).reset_index(name='
    chats')

#Apply the convert_to_json function to the chats column and
    create a new column json_string
grouped_chats['json_string'] = grouped_chats['chats'].apply(
    lambda x: convert_to_json(x))
```

A.4.2 GPT-4-Turbo interaction

Interaction with the OpenAI GPT-4-Turbo model, described in Subsection 5.3.2, has been performed using the Azure OpenAI API and writing code in Python. First of all, it has been established a connection to the Azure OpenAI API, providing the Azure OpenAI API key and endpoint, specifying the API version and the model Deployment. The core of the model interaction is the `chat_completion` function, which takes a prompt and a chat as input, introduces a delay, and sends a request to the model for chat completions using the Azure OpenAI API. The response is then processed and the completed message content is extracted and returned. The function can handle exceptions and returns an error message if encountered which reports the details of the error. The `chat_completion` function is applied to each row of the *json_ string* column in the previously prepared dataframe, *grouped_ chats*, and the tqdm library is used to visualize progress. Completions are stored in a new column named *result_ column* in the dataframe.

```python
import openai
from openai import AzureOpenAI
import time
import pandas as pd
from tqdm import tqdm
tqdm.pandas()

#Initialize AzureOpenAI client with API key and Azure endpoint
client = AzureOpenAI(
    api_key=openaikey,
    api_version="2023-10-01-preview",
    azure_endpoint=azureendpoint
)

#Specify the deployment name for the GPT-4-Turbo model
```

```
deployment_name = 'gpt4-1106-preview'

#Function for generating chat completions using GPT-4-Turbo
def chat_completion(prompt, chat):
    #Introduce a delay for better rate-limiting management
    time.sleep(1)
    try:
        #Request completion from the GPT-4 model
        response = client.chat.completions.create(
            model=deployment_name,
            response_format={"type": "json_object"},
            messages=[
                {"role": "system", "content": prompt},
                {"role": "user", "content": chat}
            ],
            max_tokens=1000,
            temperature=0
        )
        #Extract the completed message content from the response
        result = response.choices[0].message.content
        return result

    #Handle exceptions and return an error message
    except Exception as ex:
        template = "Error: {0}; {1!r}"
        message = template.format(type(ex).__name__, ex.args)
        return message

#Apply chat completion using tqdm for progress visualization
grouped_chats['result_column'] = tqdm(grouped_chats['json_string
   '].progress_apply(lambda x: chat_completion(prompt, x)))
```

A.4.3 Model output post-processing

The last part of the feature extraction process is the post-processing of the output, detailed in Subsection 5.3.3. First of all, the `parse_json_or_return_default` function has been defined, to handle JSON parsing or return a default value in case of a JSON decoding error. Indeed, the purpose of the function is not to discard completions that are not in JSON formats, including errors, but rather to store them along with the interaction ID for subsequent analysis. The function returns *True* if the parsing was successful and *False* otherwise. The function has been applied to the *result_ column* of the *grouped_ chats* using the `apply` method and the result is expanded into two new columns, *result_ column* and *success* using the `pd.Series` method. The original dataframe *grouped_ chats* has been concatenated with the normalized JSON content from the *result_ column* using `json_normalize` for flattening JSON structures. The final dataframe has been created by combining the original columns with the expanded JSON content columns.

```python
import pandas as pd
from pandas import json_normalize
import json

#Function to parse JSON or return a default value
def parse_json_or_return_default(value):
    def safe_json_loads(x):
        try:
            return json.loads(x), True
        except json.JSONDecodeError:
            return value, False
    #If the value is empty, return an empty dictionary and False
    if not value:
        return {}, False

    return safe_json_loads(value)

#Apply the parse_json_or_return_default function to handle JSON
    parsing and success detection
grouped_chats[['result_column', 'success']] = grouped_chats['
    result_column'].apply(lambda x: parse_json_or_return_default(x
    )).apply(pd.Series).rename(columns={0: 'result_column', 1: '
    success'})

#Concatenate the original data frame with the normalized JSON
    content
df_final = pd.concat([grouped_chats, json_normalize(
    grouped_chats['result_column'])], axis=1)
```

Bibliography

[Azu, a] Azure openai service documentation. <https://learn.microsoft.com/en-us/azure/ai-services/openai/>. 2nd December 2023.

[Azu, b] Azure products. <https://azure.microsoft.com/en-us/products>. 2nd December 2023.

[Azu, c] Cloud computing terms. <https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/>. 2nd December 2023.

[Azu, d] Microsoft azure data factory documentation. <https://learn.microsoft.com/en-us/azure/data-factory/>. 14th December 2023.

[Azu, e] What is azure databricks? <https://learn.microsoft.com/en-us/azure/databricks/introduction/>. 2nd December 2023.

[Ope, 2024] (2024). Prompt engineering guide. <https://platform.openai.com/docs/guides/prompt-engineering>. 9th Jenauary 2024.

[Al-Jumaili et al., 2023] Al-Jumaili, A. H. A., Muniyandi, R. C., Hasan, M. K., Paw, J. K. S., and Singh, M. J. (2023). Big data analytics using cloud computing based frameworks for power management systems: Status, constraints, and future recommendations. *Sensors*, 23(6):2952.

[Berisha et al., 2022] Berisha, B., Mëziu, E., and Shabani, I. (2022). Big data analytics in cloud computing: an overview. *Journal of Cloud Computing*, 11(1):24.

[Bose, 2009] Bose, R. (2009). Advanced analytics: opportunities and challenges. *Industrial Management & Data Systems*, 109(2):155–172.

[Brown et al., 2020] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

[Bsharat et al., 2023] Bsharat, S. M., Myrzakhan, A., and Shen, Z. (2023). Principled instructions are all you need for questioning llama-1/2, gpt-3.5/4. *arXiv preprint arXiv:2312.16171*.

[Cloud, 2011] Cloud, H. (2011). The nist definition of cloud computing. *National Institute of Science and Technology, Special Publication*, 800(2011):145.

[Collier and Shahan, 2015] Collier, M. and Shahan, R. (2015). *Microsoft Azure Essentials-Fundamentals of Azure*. Microsoft Press.

[Devlin et al., 2018] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

[Dixon, 2010] Dixon, J. (2010). Pentaho, hadoop, and data lakes. `<https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/>`. 19th November 2023.

[Ekin, 2023] Ekin, S. (2023). Prompt engineering for chatgpt: A quick guide to techniques, tips, and best practices.

[Elgendy and Elragal, 2014] Elgendy, N. and Elragal, A. (2014). Big data analytics: A literature review paper. volume 8557, pages 214–227.

[García-Peñalvo and Vázquez-Ingelmo, 2023] García-Peñalvo, F. and Vázquez-Ingelmo, A. (2023). What do we mean by genai? a systematic mapping of the evolution, trends, and techniques involved in generative ai. *International Journal of Interactive Multimedia and Artificial Intelligence*, In Press.

[Hadi et al., 2023] Hadi, M. U., Qureshi, R., Shah, A., Irfan, M., Zafar, A., Shaikh, M., Akhtar, N., Wu, J., and Mirjalili, S. (2023). A survey on large language models: Applications, challenges, limitations, and practical usage. *TechRxiv*.

[Huang et al., 2023] Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., Chen, Q., Peng, W., Feng, X., Qin, B., et al. (2023). A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *arXiv preprint arXiv:2311.05232*.

[Inmon, 2002] Inmon, W. H. (2002). *Building the Data Warehouse,3rd Edition*. John Wiley & Sons, Inc., USA, 3rd edition.

[Kao et al., 2009] Kao, E. C.-C., Liu, C.-C., Yang, T.-H., Hsieh, C.-T., and Soo, V.-W. (2009). Towards text-based emotion detection a survey and possible improvements. In *2009 International conference on information management and engineering*, pages 70–74. IEEE.

[Kojima et al.,] Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. Large language models are zero-shot reasoners, 2022. *URL https://arxiv.org/abs/2205.11916*.

[Lewis et al., 2020] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

[Liddy, 2001] Liddy, E. D. (2001). Natural language processing.

[Liu et al., 2023] Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., and Neubig, G. (2023). Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35.

[Microsoft, a] Microsoft. Advanced analytics architecture. <https://learn.microsoft.com/en-us/azure/architecture/solution-ideas/articles/advanced-analytics-on-big-data>. 22th December 2023.

[Microsoft, b] Microsoft. Modern analytics architecture with azure databricks. 19th December 2023.

[Microsoft, c] Microsoft. What is a data lake? <https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-a-data-lake>. 19th November 2023.

[Mishra and Misra, 2017] Mishra, S. and Misra, A. (2017). Structured and unstructured big data analytics. In *2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC)*, pages 740–746.

[Nambiar and Mundra, 2022] Nambiar, A. and Mundra, D. (2022). An overview of data warehouse and data lake in modern enterprise data management. *Big Data and Cognitive Computing*, 6(4).

[Nandwani and Verma, 2021] Nandwani, P. and Verma, R. (2021). A review on sentiment analysis and emotion detection from text. *Social Network Analysis and Mining*, 11(1):81.

[OpenAI, 2023a] OpenAI (2023a). Gpt-4 technical report.

[OpenAI, 2023b] OpenAI (2023b). OpenAI website. <https://openai.com>. 19th November 2023.

[Parrott, 2000] Parrott, W. G. (2000). Emotions in social psychology: Key readings.

[Radford et al., 2018] Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. (2018). Improving language understanding by generative pre-training.

[Radford et al., 2019] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

[Raffel et al., 2020] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

[Wang et al., 2022] Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. (2022). Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171.*

[Wei et al., 2022] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

[Xu et al., 2023] Xu, B., Yang, A., Lin, J., Wang, Q., Zhou, C., Zhang, Y., and Mao, Z. (2023). Expertprompting: Instructing large language models to be distinguished experts. *arXiv preprint arXiv:2305.14688.*

[Yang et al., 2023] Yang, J., Jin, H., Tang, R., Han, X., Feng, Q., Jiang, H., Yin, B., and Hu, X. (2023). Harnessing the power of llms in practice: A survey on chatgpt and beyond. *arXiv preprint arXiv:2304.13712.*

[Zhao et al., 2023] Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., et al. (2023). A survey of large language models. *arXiv preprint arXiv:2303.18223.*