POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Conversational Access to Structured Knowledge exploiting Large Models

Tesi di Laurea Magistrale in
Computer Science and Engineering

Author: **Vincenzo Manto**

Student ID: 990165
Advisor: Prof. Maristella Matera
Co-advisors: Emanuele Pucci
Academic Year: 2022-23

Abstract

Data and information are the heart of the decision-making process. Information needs require the adoption of innovative technologies and a deeper approach to the object of investigation. The expansion of conversational intelligence over the last decade makes natural language systems a great tool, allowing the user to use voice or text channels to extract and process data.

This Thesis proposes an approach for the design of an infrastructure aimed at the extraction, processing, presentation and visualization of structured data, in light of the developments of conversational agents, based on the simplicity of use and on the inference capabilities of the model.

The approach exploits a series of elaborations, translations and modeling that allow inexperienced business users to connect a relational database, interact with the data and extract information and knowledge from the results by dialogues. A great contribution to this process is given by large models that represent an important new frontier of conversational intelligence.

More specifically, this Thesis focuses on the following contributions:

- A characterization of the information exploration platform, in terms of functional and non-functional requirements and objectives;
- A methodology to connect data sources through minimal annotations, explore them through large models, expose the results and manipulate them in order to generate new knowledge thanks to data summarization and data visualization, making possible a deeper exploration;
- An architecture for a framework that, by exploiting state-of-the-art technologies, supports the distribution of a flexible and modular platform for accessing and processing information;
- A prototype of the framework that integrates different technologies.

The Thesis exploits and reports studies on business users that compared the performance and user satisfaction with regard to this proposal.

Sommario

I dati e l'informazione sono il cuore del processo decisionale. Le esigenze informative richiedono l'adozione di tecnologie innovative ed un avvicinamento dell'utente all'oggetto di indagine. L'espansione dell'ultimo decennio in materia di intelligenza conversazionale rende i sistemi a linguaggio naturale un ottimo alleato, permettendo all'utente di utilizzare canali vocali o testuali per estrarre e elaborare dati.

Questa Tesi propone un approccio per la progettazione di un'infrastruttura finalizzata all'estrazione, elaborazione, presentazione e visualizzazione di dati strutturati, alla luce degli sviluppi sugli agent conversazionali e ponendo come fondamento la semplicità di utilizzo e le capacità inferenziali del modello.

L'approccio sfrutta un susseguirsi di elaborazioni, traduzioni e modellazioni che consentono a utenti aziendali inesperti di collegare un database relazionale, interagire con i dati ed estrarre informazioni e conoscenza dai risultati tramite dialoghi. Un grande contributo a questo processo è dato dai large models che rappresentano una nuova importante frontiera dell'intelligenza conversazionale. Questo elaborato verte sui seguenti contributi:

- Una caratterizzazione della piattaforma per l'esplorazione dell'informazione, in termini di requisiti e obiettivi funzionali e non funzionali;
- Una metodologia per collegare sorgenti di dati tramite annotazioni minimali, esplorarle tramite large models, esporre i risultati e manipolarli al fine di generare nuova conoscenza grazie a data summarization e data visualization, rendendo possibile una più profonda esplorazione;
- Un'architettura per un framework che, sfruttando tecnologie all'avanguardia, supporti la distribuzione di una piattaforma flessibile e modulare di accesso all'informazione;
- Un prototipo del framework che integra diverse tecnologie.

La tesi sfrutta e riporta studi su utenti business valutando la loro soddisfazione nei riguardi di tale proposta.

Contents

Ringraziamenti

Grazie a tutti

1 | Introduction

In today's data-driven world, accessing and using data is becoming increasingly important for the entire society, spanning a wide range of industries.

Retracing the history of information technologies, the centrality of unstructured information becomes clear: approximately 80% of digitized information is expressed, accessed and held in an unstructured format [1]. With the advent of the Web, structured has shared the ground with semi-structured or completely unstructured formats such as multimedia content. However, the role of structured information is still cardinal today, being at the base of the engines and systems that support entire sectors.

Nonetheless, the development of traditional methods of accessing and analyzing structured data has found it very difficult to abandon the technical approach and has crystallized on highly structured tools, such as the by-now standard relational database or SQL. Such methodologies require specialized technical knowledge and can be time-consuming and error-prone, reducing direct access to data to a very elitist practice. As a result, there is a growing need for more intuitive and easy-to-use ways to access structured data that don't require training or specialized skills.

The now fertile ground of artificial intelligence technologies seems to have paved the way for more natural and therefore "democratic" access to structured data. In this scenario, we are experiencing a sharp increase in the role of chatbots, becoming prevalent across a wide range of activities.

This tremendous growth is due to the disruption in information exchange induced by chatbots, lead by the simplified interaction between users and applications using natural language. This trend makes conversational agents one of the most promising technologies to transform software platforms into conversational systems. Finally, the highlighted trend has a cross-cutting impact that involves technical innovation and effort as well as a deep change of meaning related to information-gathering methods.

1.1. Scope framing

Today's world is largely based on production and business processes. The large use, consumption and generation of information derive in fact from the landscape of companies and their productive activities. In analyzing IT models and methods that can be useful and usable in today's society, we cannot fail to take into account the needs and desires of the productive sector. Therefore, in this Thesis, we try to examine problems and solutions from the perspective of business decision-makers, large consumers of both aggregate and raw data, primary users and early adopters of these new technologies.

Furthermore, the focus of this Thesis is on the widespread relational approach. However, recognizing the role of non-relational structures, efforts have been made to at least include noSQL environments such as associative databases.

To better frame the problem scope, we propose a real user case: imagine a logistics manager, interested in verifying the lead times between invoicing and shipping, thus considering any status changes in printing and packaging. The main entity of this extraction is the ASN or shipping document. To extrapolate this data twice a day and to reduce costs, the IT manager chooses to materialize and directly query the transactional database while maintaining a relational approach. The manager's knowledge about the database is scarce and there is no technical expertise in the extrapolation and manipulation of data. To date, access to this information is delegated to business intelligence (BI) infrastructures and dashboards generated when necessary by specialized technicians.

For simplicity, we can assume that the ASNs are documents and that the shipping information is recorded in a specific table linked to a single document. Generally, managers are not required to have extensive and precise knowledge of every entity, but we can assume that each of them knows processes and their key elements.

From interviews and practical experiences, the problem to be addressed can be functionally divided as follows:

1. Allowing the system to connect to the data source and to join abstract concepts - e.g. entities - to table structures. This step must be simple and intuitive.

2. Being able to translate a need *Give me the delivery lead time of the ASNs generated in the last 6 hours group by shipping airport* into an executable SQL query. This step requires acknowledging the entities involved and requests and then translating the prompt into an executable command on the data. Moreover, this request, like many different ones, can be expressed in various ways by presenting different entities, values and conversational objects.

   The query could be (MSSQL):

```
SELECT
    S.DepAirport as Airport,
    avg(S.FlightDate - D.InvoiceDate) as Leadtime
FROM Document D LEFT JOIN ShippingDet S
    ON S.DocNum = D.Id
WHERE
    D.InvoiceDate >= GETDATE() - 21600
GROUP BY S.DepAirport
HAVING S.FlightDate IS NOT NULL
```

   Understandably, this query cannot be written without deep knowledge of tables, SQL and the logic expressed by the DBMS.

3. Extracting the data and presenting it. The data format must be understandable and consistent with the request. Referring to the presented end user, the data should be summarized and explained.

4. Once extracted, the data must generate information and knowledge. Thus, data transformation should not be delegated at a later time but performed simultaneously with the request. The results must be interpretable and this objective is achieved by including **data summarization** and **data visualization**.

   The data should be visualized through graphs at the user's request, but new information must be generated through a descriptive and predictive approach.

5. Allowing the user to move and pivot between different entities and related contexts, also autonomously manipulating the visualization based on his/her needs.

This quick example demonstrates the breadth of the problem under consideration. However, unlike the previous works [2] [3] on which this Thesis is based, the center of the scope appears shifted: indeed, if until now great efforts were aimed at directly translating natural language into structured queries, the introduction of some groundbreaking and outperforming products may discourage further developments in this direct field since they can obtain better qualitative performances at a negligible cost.

This Thesis follows the assumption that conversational access to data is not just about translating the natural text into commands, but it concerns the entire infrastructure around it, generating a modular yet integrated data-analysis environment. There is a need to encapsulate natural query language (NQL) engines within platforms that facilitate data connection, but are also capable of integrating wider natural language understanding (**NLU**), **schema pruning**, **data mining**, **data visualization** and **data summarization** to support information gathering and elaboration, suggesting insights as well as supporting the decision-making process.

This is how conversational access to structured data comes into play. In this Thesis, we refer extensively to **NQL** as a form of natural language processing (NLP) that allows users to interact with structured data using natural language queries, such as *Show me all the customers who made a purchase last month.* However, conversational access to structured data goes one step further, allowing users to dialogue with the system, asking also follow-up questions about information extraction, manipulation, analysis and elaboration.

Taking a user-centric viewpoint, simplifying the accessibility and analysis of structured data for non-technical users has the potential to enhance productivity for end users, specifically business managers in this Thesis. It is not debatable that this innovation may also have a social impact by democratizing access to information, making data extraction and consumption easier for employees at all levels of an organization.

Nevertheless, despite the advantages mentioned above, there are still numerous challenges involved in the successful implementation of conversational and chat-based data gathering, ranging from developing precise and efficient algorithms to ensuring stringent data privacy measures. Notwithstanding, one of the key issues to overcome is the integration of these new technologies into

user interfaces that are intuitive and facilitate seamless interaction with the system. It is crucial to not only focus on providing functional access to information but also to foster innovation by extracting knowledge and insights from data.

The future approach towards information would probably not only be a matter of extracting but also communicating, transferring and manipulating useful data to transform it into knowledge and wisdom.

Indeed, the potential of conversational data access will be explored and the practical capabilities of these presented technologies will be demonstrated, also highlighting the challenges involved in effectively implementing these technologies by reviewing the existing literature on conversational interfaces and developing a prototype system that combines and bundles different tools.

This work aims to better understand the opportunities and challenges involved in implementing a *holistic* conversational platform to access, communicate, manipulate and analyze structured data and provide further insights.

The felt importance of this new field can be decomposed into several reasons.

First, with organizations placing growing reliance on structured data for informed decision-making and gaining a competitive edge, the need for intuitive and user-friendly approaches to access and analyze this data becomes paramount. Indeed, traditional methods, such as SQL queries or data analysis through Excel, require specialized technical expertise and can be a long error-prone process. Nonetheless, conversational access to data could provide a more accessible and efficient way for non-technical users to interact with data.

Second, the potential benefits of conversational access to structured data are significant and organizationally relevant. By democratizing access to data and making it easier for users - at all levels of an organization - to leverage data in their activities, these technologies can increase productivity, by reducing errors and helping organizations make more informed decisions.

Last but not least, conversational access is technologically important. The development of efficient NLP algorithms, combined with intuitive user interfaces, can make the interaction with data analysis systems easier for users, innovating but also disrupting the way we are used to collecting and managing knowledge.

Thus, research into these technologies can help drive innovation and advance the state-of-the-art in natural language processing, conversational interfaces, and data access systems more broadly.

1.2. Definitions

Chatbot
is a conversational software agent capable of natural language processing to simulate human conversation, either through text-based or voice-based communication.

Schema Annotation
is the process of characterizing tables, relations, and attributes of a database conceptual model to produce a mapping between the data source and the conversation interaction basis.

Data Visualization
is the process of representation of data in a visual form such as charts or graphics, making access to information easier and precognitive.

Data Transformation
is the process of converting data from one format or type to another, to make it more useful for a particular application.

Conversational Context
refers to the background information that is used to interpret and understand a conversation. In a chatbot, the conversational context could be referred to contextual information or previous messages.

Chat Interface
is a system that allows users to interact with a chatbot through text-based communication.

NLP
acronym for Natural Language Processing, is the AI process of interpreting and generating human language, which applications are often used in chatbots to help them understand and respond to user requests.

Pretrained Model
is an AI model that has been trained on a large dataset and is available for use without additional training, making them extremely scalable and extensible

Data Summarization
is the process of reducing large datasets into smaller, more manageable summaries or representations that retain the most important information, helping readers to better access and merge information - e.g. trends, patterns, and outliers in the data.

GPT Models Family
is a series of language models developed by OpenAI, which use deep learning techniques to generate human-like text. These models have been trained on massive datasets and are capable of a wide range of language tasks, including language translation, text completion, and question answering.

LLMs
or Large Language Models, refer to a class of artificial intelligence models that combine natural language processing techniques with logic-based reasoning to enable machines to reason and infer based on natural language input.

LLM models are designed to understand natural language text in a more structured and precise way than traditional machine learning models, which enables them to perform more complex tasks such as answering complex questions, solving problems, and even generating new text.

3-Tier Architecture
is a model of IT infrastructure that separates the application layer - or business logic layer - and the database layer. It is composed of a presentation tier - also called client or front-end -, an application tier - also called the back-end or application layer - and a database tier.

DBMS
Database Management System (DBMS) is a software application used to populate and manage data in an organized and structured way.

SQL
acronym for Structured Query Language, is a programming language specifically designed for managing and querying relational databases

API
acronym for Application Program Interface, is a set of protocols that specify how software components should interact with each other.

OLTP

or Online Transaction Processing is a type of database management system that is optimized for handling large numbers of short transactions, such as updating or deleting small amounts of data in real-time. These systems are designed to ensure data consistency and employ ACID (Atomicity, Consistency, Isolation, and Durability) properties, using a normalized data model, and are optimized for read-and-write tasks.

2 | State of the art

In the era of technological innovation, the role of automation, interaction and accessibility continues to drive the advancement of many IT sectors. In this large and captivating landscape, chatbots and conversational agents represent some of the crucial actors of this epochal change.

The field of conversational intelligence, specifically chatbots, has undergone a significant disruption in recent years with the emergence of several conversational models. This breakthrough has drastically changed the landscape of the field, making it difficult to snapshot the current state of the art due to the rapid evolution of research and the widespread adoption of pre-trained models. The big bang disruption caused by GPT has led to the development of more sophisticated and accurate chatbots, making it difficult to keep track of the latest developments. Despite these challenges, this chapter aims to provide an overview of the consolidated state of the art in conversational intelligence as of March 2023.

At the end of 2022, the technological landscape was enriched with great tools, thanks to the commitment of companies and organizations working to improve the capabilities and functionality of these systems. Current developments in the field focus on improving the performance of chatbots and NLP capabilities to allow chatbots to understand more complex and nuanced input. Finally, a further essential point of research is the development of user-friendly interfaces to make chatbots more widely accessible in a variety of domains.

2.1. Conversational AI: technological landscape

The efforts made lately involve many organizations, but it owes part of its explosive evolution to the existence of some private companies known for the development of advanced conversational AI technologies, some of them highly renowned:

- **OpenAI** is a research organization that focuses on the development of artificial intelligence and NLP technologies. Its Generative Pretrained Transformer (GPT) model family is a prime example of advanced conversational intelligence and is capable of generating human-like text on a wide range of topics and styles.
- **Google** is a leading developer of conversational intelligence technologies, including its Google Assistant platform, Google Bart and its Dialogflow chatbot development platform. These tools enable developers to build intelligent chatbots and other conversational interfaces that can understand and respond to natural language input.
- **Microsoft** is a global technology company that has developed a range of conversational intelligence platforms and tools, including the Azure Bot Service and the Language Understanding Service. Extremely tied up with OpenAI in developing GPT models and their applications - e.g. BingAI -, Microsoft has deserved an important mention in this area. These tools enable developers to build intelligent chatbots and other conversational interfaces that can understand and respond to natural language input.

More generally, the state of the art in conversational bots is constantly evolving. the technological landscape is teeming with relevant projects and products such as BlenderBot, built and open-sourced by Facebook AI. "It is the largest-ever open-domain chatbot and outperforms others in terms of engagement and also feels more human, according to human evaluators"[4]. It is the first chatbot to blend a diverse set of conversational skills — including empathy, knowledge, and personality — in one system [4].

Another approach to building state-of-the-art conversational AI is through transfer learning and using large-scale language models like OpenAI GPT3. These techniques allow for the creation of powerful conversational AI within hours. Thus, the constantly evolving state of the art in conversational bots is pushing the development of more advanced and sophisticated conversational AI tools that can offer more personalized and engaging interactions with users.

Contemporary conversational agents primarily rely on Natural Language Processing (NLP) methods that combine linguistic analysis and artificial intelligence. These approaches enable computers to comprehend and analyze human language, facilitating various subsequent tasks.

NLP can be broadly divided into two components: Natural Language Understanding (NLU), which focuses on interpreting the user's input, and Natural Language Generation (NLG), which involves generating appropriate responses [5].

In the realm of business, chatbots are increasingly utilized to automate customer service tasks and enhance customer engagement. They can be programmed to recognize specific keywords or trained using machine learning to provide more natural responses. As a result, chatbots can assist businesses in generating sales, automating customer service processes, and executing various tasks [6].

Also at the business level, wide is the usage of OpenAI GPT, which represents a state-of-the-art language processing AI model developed by OpenAI. This model is capable of generating human-like text and has a wide range of applications, including language translation, language modeling, and text generation for applications such as chatbots, but also text completion and analysis.

GPT works by processing input in natural language using a neural network. It is trained on a sizable dataset of text, including books, articles, and other textual content. The latest version of GPT is GPT-4 which is a large multimodal model that accepts image and text inputs and emits text outputs. "While it is less capable than humans in many real-world scenarios, it exhibits human-level performance on various professional and academic benchmarks" [7]

One key aspiration of AI is to develop natural and effective task-oriented conversational systems that use a natural language interface to extract data from structured sources, supporting people in accomplishing specific goals and activities. These systems go beyond chitchat conversation and can be used for tasks such as generating a sales report from a database or easing the stress of trip planning [8].

Another example of such a system is the Grounded Open Dialogue Language Model (GODEL), a task-oriented conversational system developed by Microsoft Research. GODEL achieves new state-of-the-art performance in language-driven data exploration by using techniques such as pre-training for context representation in conversational semantic parsing.

Another approach to conversational access to data is through Conversational Text-to-SQL (CoSQL) tasks that map natural language utterances in a dialogue to SQL queries. State-of-the-art systems use large, pre-trained and fine-tuned language models, such as the T5-family, in conjunction with constrained decoding [9].

Alongside conversational access to structured data, another important and evolving topic is the automated conversation-based generation of dashboards, that is to say, advanced visual extraction of data. One approach for the automated generation of engaging dashboards is presented by researchers at Microsoft. "Their approach employs a decision model for visualizing Key Performance Indicators (KPIs) that are developed based on dashboard design principles in the literature" [10]. The decision model is used to automatically generate engaging dashboards for organizations, constituting a highly appreciated tool.

Microsoft's decision model takes into account various attributes of KPIs, such as whether a KPI has a single value or a set of values, and the purpose of the KPI - e.g., to reveal relationships between values. Based on these attributes, the decision model determines how a KPI should be visualized using common types of visualization elements such as tables and graphs.

2.2. Chatbots

Chatbots have come a long way in recent years and their evolution is deeply linked with their development patterns and technologies. More specifically, focusing on the technical aspects of conversational intelligence, design approaches can be broadly classified into two main categories: rule-based and self-learning, even if more precise classifications exist and the field is living a continuous evolution.

In a rule-based approach, the chatbot is designed to respond to user inputs based on a set of predefined rules that may involve matching certain keywords or phrases in the user's input to a pre-determined response, thus, limiting its serendipity and adaptability capabilities. For instance, a rule-based chatbot might be programmed to respond to the keyword `hello` with the response

> *Hello! How can I help you?.*

Even if rule-based chatbots can be effective for certain use cases - e.g. task completion -, their responses are limited to the rules set by their developers and they cannot learn from new user inputs.

Self-learning - i.e. data-driven approaches - chatbots, on the other hand, use machine learning algorithms to improve their performance over time, learning from large amounts of data, such as transcripts of human conversations or text corpora.

This class of chatbots can be further classified into Retrieval Based and Generative. Retrieval Based chatbots use a repository of predefined responses and select the most appropriate response based on the user's input. Generative chatbots, on the other hand, generate responses from scratch using natural language generation techniques.

GPT

Being one of the core parts of the proposed prototype and one of the most important natural language models, it is worth describing the GPT model family. More specifically, the GPT model family is a group of natural language processing models that use deep neural networks to generate coherent and fluent text.

These models are different from rule-based in several ways. Rule-based chatbots rely on predefined rules and scripts to respond to user queries. GPT models and ChatGPT specifically, on the other hand, do not need any specific rule but can generate diverse and natural responses based on the input text, relying on human feedback reinforcement learning (RLHF). Indeed, they can entirely leverage deep learning techniques and transformer architecture to produce text that closely resembles human writing.

They are trained on large amounts of unlabeled data, such as web pages, books, and news articles, and then fine-tuned for specific tasks, such as question answering, sentiment analysis, and language translation.

To generate output texts, GPT models use a technique called autoregressive generation, which means they predict the next word or token based on the previous ones [11]. The output texts can be controlled by using different parameters, such as temperature, top-k, and top-p, which affect the randomness and diversity of the generated texts.

From a development perspective, producing chatbots can be extremely challenging when starting from scratch. For this reason, several platforms and tools have emerged over the past decade to help developers build chatbots without coding, such as, for example, visual drag-and-drop bot editors, which allow developers to create streams of conversations simply and intuitively. Notwithstanding, many other developers generally prefer to use chatbot platforms like Google Dialogflow [12], Amazon Lex [13], IBM Watson Assistant [14], Wit.ai [15] and Microsoft Azure Bot Service [16], which provide tools and resources to build and deploy more complex but efficient chatbots.

Despite this, the process of creating a chatbot usually involves several intricate steps. In general, the procedures consist in identifying the chatbot's purpose and then determining the final interface of the product, for example, web-distributed. Next, developers have to choose the chatbot platform that best suits their needs and design the conversation flow in a chatbot editor. Developers must then test their chatbot to ensure it works correctly before training it using machine learning techniques if applicable. Collecting feedback from users is crucial to improve the chatbot's performance, and developers should monitor chatbot analytics to track its performance over time.

2.2.1. Chat interface

As already said, when developing a chatbot, it is decisive to assess the communication medium through which users will interact with the system.

One of the most adopted solutions is to develop a customized chat interface that can be distributed locally or as a remote service, such as exploiting a web page or a Progressive Web App (PWA). However, this choice requires some extra effort from the developer, including introducing dedicated buttons to improve the quality of the conversation.

Alternatively, some developers prefer to leverage existing chat platforms that provide an API to control the messages sent and received by the online profile. These platforms are always available and offer advanced communication features, such as buttons, image exchange, and document sharing. By using multiple platforms simultaneously, developers can communicate with their APIs without changing the application logic.

This choice becomes even more crucial considering the increasing request for chat-based services delivered through virtual assistants such as Alexa or Google Assistant, introducing a collateral issue about pure vocal agents.

Finally, presenting structured data within a chat requires considering several factors to ensure the user can easily interact with the results and subsequentially extract usable insights from them. Indeed, structured data can be presented employing tables, lists, charts, and graphs, but considering that the presentation must be optimized to provide a seamless user experience. Nevertheless, commonly used visualization methods may be too complex to display in a chat window or conversationally summarized.

Thus, it is not just a matter of accessibility but a complete and still-developing revision of the interaction.

2.3. Data extraction and presentation

In the world of commercial companies, data retrieval is a crucial step in collecting data from various sources for processing and analysis. In these processes, extraction can be performed through several approaches, such as web scraping, Extract-Transfer-Load (ETL) methods, or using APIs to fetch data from external sources. Data are then processed to extract information to be used to create dashboards and visualizations, providing real-time insights into business performance.

When talking about operative data, most business information and knowledge are integrated into comprehensive systems called Enterprise Resource Planning (ERP) systems. Such systems contain a large amount of data that can be used to generate KPIs and dashboards. Many modern ERP systems come with built-in reporting and analytics capabilities, making it easy for users to generate KPIs and dashboards without needing technical skills. However, if a user needs to create new extractions, it would be necessary to use SQL or other tools to fetch data from the ERP system for analysis. Querying and preparing the data is typically performed by a data analyst or other technical staff member experienced in working with SQL and particular ERP systems.

Depending on the specific requirements of a company, software engineers and DB managers can opt for data warehousing or operative data analysis as two possible ways of managing information for analysis.

A data warehouse is a centralized repository of historical data extracted, transformed and loaded from various sources. Warehousing is generally used when the company requires large analytical queries and reporting for decision-making.

On the other hand, an operative data analysis on transactional databases is a real-time or near-real-time analysis of operational data generated by business processes. Contrarily to data warehousing, it commonly supports operational tasks and performance monitoring for business optimization.

Once the data have been extracted, it is usually analyzed using Business Intelligence (BI) tools to generate KPIs and dashboards. These tools allow users to create interactive visualizations, providing real-time insight into business performance. Generally speaking, BI tools are used in conjunction with ERP systems exploiting specific extraction, materialization, and transfer methodologies.

Significant advancements have been made in the realm of data storytelling and data visualization. Various techniques are available to present information effectively, including the utilization of tools such as Tableau [17], which enables the creation of interactive charts and graphs. By leveraging these tools, users can easily manipulate data fields and design visual representations, providing valuable insights into company performance.

Up to March 2023, conversational access to data and the visualization of results have also improved. Data extraction languages have become less technical, such as the KQL used by the ElasticSearch-Kibana [18] duo, representing a popular combination for searching and analyzing data. PowerBI [19], a Microsoft product specialized in BI, provides integrated conversational access, enabling users to efficiently extract queries. However, there is still ample room for research and experimentation in these fields and on the consequent human-computer interaction.

2.3.1. Traditional business data extraction

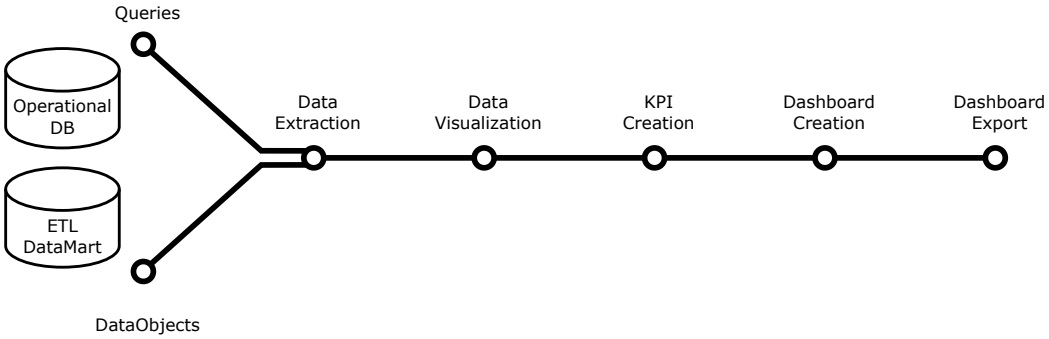


Figure 2.1: A traditional business process of data extraction

Extracting business data from an ERP system can seem daunting to non-expert users. To better understand how user-data interaction is traditionally handled, it may be convenient to highlight a summarized yet common business process, following Figure 2.1.

The process is typically initiated with SQL queries to retrieve the data from the ERP database. Commonly data need to be transformed for analysis, a task which can be automated by ETL tools (Extract, Transform, Load) and then loaded into a data warehouse. However, it is not uncommon to directly access operational resources to get a real-time perspective. In some cases, reporting tools can allow users to access information employing DataObjects - a higher level of abstraction over data schema -: this methodology eases the data extraction, however, it may limit the user to a smaller set of entities and operations.

Once extracted and elaborated, data can be visualized with dashboard and reporting tools providing an intuitive interface for non-technical users to create interactive dashboards and reports. Users can customize charts and graphs with drag-and-drop data fields, filters, and visualization types to gain insights into business performance.

Alternatively, some ERP systems come with built-in reporting tools, which can extract data directly from their system. These tools offer pre-built reports, specifically designed for certain business processes, and can be adjusted to meet specific needs. The reports can be generated on-demand or scheduled and can be exported in various formats such as PDF and CSV.

In conclusion, while extracting business data from an ERP system can present complexities, leveraging specialized tools and technologies can make it accessible even to non-expert users. Indeed, visualizing and analyzing business data play a crucial role in making informed decisions, and integrated ERP systems offer robust tools to facilitate this objective.

2.4. Related research approaches

Some recent studies have proposed novel approaches to tackle the problem of conversational access to data and information, demonstrating a strong commitment of the academic world to this topic. This Thesis finds its basis in the article "Conversational Data Access"[2] and the Master Thesis "A conceptual modeling approach for the rapid development of chatbots for conversational data exploration"[3], which aim at defining a consolidated framework for data-driven chatbots. This approach follows an intent-entity paradigm for prompt-to-SQL translation, representing the starting point for this Thesis.

However, our proposal departs from a purely SQL-based approach and tries to outline a comprehensive infrastructure that integrates conversational data exploration with data analysis, data visualization, and dashboard creation. It is also important to consider the development (prototyping) of data-driven chatbots for accessing structured data. At the academic level, many efforts are made in this area, such as "Rapid prototyping of chatbots for data exploration"[20] which reports on the experience of defining a model-based technique for the automatic generation of chatbots for data exploration. This technique has been integrated into a no-code platform called CODEX, which offers visual notations to index relational data sources and connect them to conversation flows for data exploration.

An essential aspect to consider is adopting a conversational approach when exploring data. A significant contribution in this field is Castaldo's framework, which provides a structured framework for designing chatbots specifically tailored for data exploration. Distinct from conversational virtual assistants like Amazon Alexa or Apple Siri, this class of chatbots utilizes structured input to retrieve data from known data sources. The approach is grounded in a conceptual representation of available data sources and employs modeling abstractions that enable designers to define the role played by key data elements in handling user requests.

Leveraging the resulting specifications, the framework generates a conversation flow to explore the content provided by the designated data sources. However, unlike these works, the effort recounted in the following chapters tries to generate architectures that, using innovative technologies of the last year, can support agnostic access to data from a more global perspective, i.e., that combines data summarization, visualization, access, and presentation. In addition, the main focus is on business end users: unlike SQL experts, managers require fewer technicalities, more assistance, and less specific knowledge. This means a greater degree of delegation to the proposed architecture, and this is the challenge we face.

The research environment continues to work in this direction as well. Another noteworthy attempt[21] is the production of CAT, a tool that can be used to easily create conversational agents for transactional databases. The main idea is that for a given OLTP database, CAT uses weak supervision to synthesize the training data needed to train a state-of-the-art conversational agent, allowing users to interact with the OLTP database. Furthermore, CAT provides a ready-to-use integration of the resulting agent with the desired data source.

As the main difference from existing conversational agents, the agents synthesized by CAT are data-aware. This means that the agent decides which information to consider based on the current data distributions in the database, which usually results in significantly more efficient dialogues than data-unaware agents.

Again, unlike the proposed project, a supervised model is used to process input to a conversational agent. This approach increases the accuracy of results on a given OLTP but requires training phases and does not use inferential power to determine data visualization and presentation, a crucial factor in a modern data analysis system.

Finally, similarly to what is proposed in the paper [22], there is a field of research that utilizes an instruction-based LLM as an NQL translator: GPT-based CodexDB. This model is an SQL processing engine whose internals can be customized through natural language instructions. CodexDB is based on the GPT-3 Codex model from OpenAI, which translates text into code by breaking down complex SQL queries into a series of simple processing steps, described in natural language.

The processing steps are enriched with user-provided instructions and database property descriptions. Codex translates the resulting text into query processing code. An initial prototype of CodexDB is capable of generating correct code for most queries in the WikiSQL benchmark and can be further customized.

This work, similarly to what is proposed, replaces machine learning-based conversational models with pretrained structures, specifically with OpenAI's GPT-3.5 model introduced in 2023. Building on this starting point, this Thesis aims to fully exploit the capabilities of GPT models in SQL translation and textual reprocessing, paying more attention to the encapsulation of processes, which allows for smoother access to information even for inexperienced users.

2.5. Technologies used

The focus of this section is on the selection of technologies that were chosen to create a system capable of meeting the requirements outlined in Chapter 3, while remaining consistent with the design patterns described in Chapter 4.

2.5.1. GPT-3.5

GPT-3.5 is a language model that was specifically trained by OpenAI for generating natural text as an extension of the original GPT-3 language model, which was primarily designed for natural language processing tasks.

In the code generation landscape, this outstanding model has quickly gained popularity and outshined other text-to-code tools due to its remarkable ability to understand natural language and generate high-quality code that closely matches the user's intent. This is achieved through its advanced deep learning architecture, which allows it to process vast amounts of training data and learn the patterns and syntax of different programming languages.

With regard to this Thesis, the choice to use this pre-trained LLM derives from its exceptional performance and versatility, which have also contributed to its overwhelming success, making it a game-changer in the field of generation of coded text.

The use of GPT-3.5 for the text-to-query task is extremely innovative, relying on very fast few-shot fine-tuning which makes it easily configurable. Its use is mediated only by calling the REST APIs exposed by OpenAI, contextually attaching everything needed to the model to generate useful answers in terms of SQL queries. This implies having to pass to GPT-3.5 the span of the schema affected by the request and obtained through schema pruning (Section 4.5.1), the user prompt and, contextually, the information necessary to obtain a specific SQL query for the DBMS concerned. This means communicating for example which specific SQL dialect to use - e.g. MySQL, MSSQL, Oracle.

These considerations drive the choice of GPT-3.5, given also the developments of the last year. Furthermore, similar assessments can also be extended to future prompt-based models that are about to become a core training modality. This means that, in the future, the GPT engine can be replaced by other prompt-based engines without altering the structure of the proposed platform, making it extremely modular and interchangeable. Furthermore, it will potentially be possible to choose to integrate other non-prompt-based NQL engines directly into the appropriate module.

However, it should be noted that, being a third-party product, its integration must take into account the economic impact and the relative dependence on third parties. More specifically, each call to the GPT-3.5 model is economically evaluated based on the number of tokens requested which are assigned a unit price in relation to the different models. Understandably, these factors become crucial when considering how to market, distribute and monetize platforms that are highly dependent on external products.

2.5.2. Embedded NLU

NLU has become a critical component of many NLP applications, including chatbots, search engines, and virtual assistants. One approach to NLU is to use deep learning techniques to train models that can understand and interpret natural language input. In this domain, a great role is played by is sentence-embedding, which involves representing each sentence as a high-dimensional vector. The SentenceTransformer library, based on transformer models like BERT, has emerged as a popular choice for generating sentence embeddings. Sentence embedding is an important key widely used to evaluate user prompt topics and consequently prune the schema to be explored.

To further enhance NLU capabilities, tools like Spacy and nlpaug have been used. More specifically, Spacy is a library for advanced natural language processing in Python, designed to help build NLP applications that can process and understand large volumes of text. It provides pre-trained models for a variety of NLP tasks, including named entity recognition (NER), part-of-speech tagging (POS), and dependency parsing. This is used in combination with SentenceTransformer to add additional layers of analysis to natural language inputs.

In contrast, nlpaug is an NLP-focused data augmentation library. It offers a straightforward and scalable approach to augmenting training data for NLP models by generating synthetic data using diverse augmentation techniques. This methodology effectively enhances the robustness and accuracy of NLU models. By integrating nlpaug with SentenceTransformer and Spacy, a potent NLU toolset is formed, which can be tailored to meet specific requirements. This combination provides ample flexibility for customization and empowers users with a comprehensive toolkit for NLU tasks.

By leveraging the capabilities of these powerful tools, developers can create NLU models that can understand and interpret natural language input with high accuracy and efficiency, leading to the creation of more effective chatbots, virtual assistants, and other NLP applications that provide a better user experience.

RASA NLU

In this work, some of the NLU tasks have been delegated to a model based on RASA NLU, demonstrating the integration possibilities of the platform. RASA NLU is a popular open-source framework for building conversational AI chatbots. It includes powerful tools for natural language understanding (NLU), such as intent classification and entity extraction.

More specifically, intents represent the user's goal or purpose behind a message. For example, if a user says *I want to order a pizza*, the intent is likely to be `order_pizza`. The training of intent matcher in RASA has been performed by defining each intent and providing examples of messages that should be matched to that intent. These examples should cover the range of ways a user can exploit to express the intent. RASA uses machine learning algorithms to match new messages to the correct intent based on the examples provided.

Another key advantage of RASA NLU is entity extraction, which is the process of identifying and extracting specific pieces of information from a user's message, such as dates, times, and locations. To train the model, the entities to be extracted have been declared and documented. For example, if you're building a flight booking chatbot, you might want to extract the departure and arrival airports, the date of the flight, and the number of passengers. During the training, the developer can define each entity and provide examples of messages that contain that entity. Similarly to the intent matcher, RASA NLU uses machine learning algorithms to extract entities from new messages based on the examples provided.

Overall, RASA NLU provides a powerful and flexible framework for building chatbots that can understand and respond to natural language messages. By simply coding the intent matcher and entity extractor, developers can train the chatbot to accurately identify user intents and extract relevant information, making the agent more useful and engaging for users. These considerations make RASA NLU an important ally for the platform.

2.5.3. NodeJS, Angular and MySQL

During the planning phase of a 3-tier application development, it is crucial to consider a diverse range of tools and technologies. Within the extensive selection available for the back-end layer, Node.js emerges as a prominent choice being one of the most appreciated server-side JavaScript environments that enable developers to build network applications that are both high-performing and scalable. This well-known technology provides an extensive collection of open-source modules and seamless integration capabilities with various tools and services.

In the context of the front-end layer, due also to its increasing diffusion, the primary tool of choice is Angular, a TypeScript-based framework specifically designed for constructing web applications. Additionally, Angular offers comprehensive support for building scalable and maintainable applications, further enhancing the development process.

In the realm of the database layer, MySQL emerges as a prevalent selection owing to its user-friendly nature, scalability, and resilience. This relational DBMS offers extensive support for various data types and incorporates essential features such as transactions, indexing, and replication.

These capabilities contribute to its appeal and position MySQL as a versatile choice for managing data effectively. Together, this *triptych* provides a powerful and versatile toolkit for developing three-tier applications. Moreover, the extensive community support and wealth of documentation available for each tool make it easy to find help and support when needed. Thus, these technologies are an excellent choice for any developer looking to build complex and dynamic applications.

Upon considering various tools and infrastructures, distinct advantages and disadvantages come to the forefront. For instance, Node.js manifest a high capacity to handle a substantial volume of requests concurrently, making it well-suited for scalable applications, but it may not be the best choice for CPU-intensive tasks.

Finally, the proposed prototype is capable of connecting and extracting data from remote or local data sources which exploit MSSQL, Oracle, MySQL or specific formats such as JSON and CSV.

3 | Conversational data access

> *"Incomplete requirement is one of the major factors of projects' failure"[23]*

With the ever-growing volume of data, there are numerous innovative approaches to information management. While attempts such as Kibana's KQL and ElasticSearch have been made to streamline the process, technological readiness has only recently caught up with the demands.

Coherently, the market for human-computer interaction technologies has seen a significant surge in consumer demand. As a result, the need for new methods of interacting with information has become increasingly important. The impact of this trend is particularly evident among corporate executives, representing some of the highest data consumers.

For this reason, it is essential to understand the needs of this demographic to determine the requirements for an effective data querying and processing framework.

The idea for this Thesis was conceived as a result of 14 interviews with industry experts, demonstrating that there is a growing need for a more efficient and effective method of accessing information. This need has arisen due to the impact of Porter's information intensity on all aspects of organizational and operational life, creating a significant disparity between conventional approaches and technological capabilities.

Moreover, some of the assumptions and the considerations made during the requirements elicitation come from direct observations of use cases during the last 3 years - in a sort of *spurious* ethnography.

User sample

In this study, we have focused on a sample of 14 middle managers from IT and manufacturing organizations, who are highly educated and aged between 28 and 68 years, as reported in Table 3.1. All the subjects in the sample are employees of large companies with more than 250 employees and more than 50 million € in revenues. Many of these figures are project managers, while others are operational or logistics managers. They are primarily interested in operational information used in corporate planning and control cycles. A subset of these users also participated in the final evaluation study (see Chapter 6).

Participation code	Age	Gender	Role
P1	68	M	Sales manager
P2	48	M	Development team leader
P3	33	M	Project manager
P4	38	M	BU manager
P5	56	M	BU manager
P6	28	F	Logistics manager
P7	54	M	IT manager
P8	59	M	Sales manager
P9	62	M	Project manager
P10	38	F	Development team leader
P11	42	M	Product owner
P12	55	F	BU manager
P13	39	F	Product owner
P14	32	M	Senior developer

Table 3.1: Questionnaire participants

Study procedure

The study for requirements elicitation followed a traditional interview-based approach. To deeply inspect the participants' needs, 14 questions were defined to their daily data consumption, technologies, and aspirations for data analysis. The complete elicitation questionnaire is reported in Appendix A.

Analysis

The data gathered through the interviews highlighted that, unlike top managers, the study participants focus on structured data that are difficult to aggregate, with high timeliness - e.g., real-time or near-real-time data. Therefore, most of the tools they use access operational sources rather than data warehouses.

Daily data volumes for this sample can reach more than 20GB, with a wide range of technical expertise, from those with extensive IT backgrounds to those who are non-experts but skilled in analyzing data.

To systematically extract requirements-relevant information, a thematic analysis was applied to the transcribed interviews. More precisely, thematic analysis is a technique employed to identify and examine patterns or themes within qualitative data. This method proves valuable in extracting pertinent information from requirement engineering documents, including interviews, surveys, or user feedback.

By applying this approach, software engineers can gain insights into stakeholders' needs, preferences, and expectations, enabling them to prioritize and validate system requirements effectively.

One tool that supports thematic analysis in software engineering requirements elicitation is ELICA (Elicitation Aid Tool)[24]. ELICA is an automated tool capable of dynamically extracting text snippets from elicitation documents. These snippets contain information relevant to requirements, encompassing aspects such as goals, features, constraints, and quality attributes.

In our study, the ELICA tool has been partially applied to extract requirements-relevant tokens and keywords, by using a hand-made version using Python. Initially, the interviews have been collected and reorganized. Once ready, each answer has been processed by `yake` Python library to extract relevant keywords and group them by synonyms.

Therefore, the result of the interviews appears as a dictionary of relevant keywords with related frequency for each question. These outcomes allow for the classification and extraction of only requirements-relevant information reducing the noise of open answers. These results have been then analyzed and summarized in the following main requirements.

3.1. Main results

To outline the system's main objectives, it is necessary to specify the expected goals highlighted by the interview elicitation.

This proposal aims to offer a systematic method that enables the user to take control over the access, extraction, and display of information, without the need for technical or theoretical knowledge of the underlying infrastructure. Throughout the entire Thesis, the main focus is on middle managers as key users of our proposal.

The core elicitated requirement is the agnosticism towards the queried data source and the user's technical knowledge, meaning that users are not required to have an in-depth understanding of the application or connected data source. Linked to this element, another key point is usability: the expected platform's interactions must be designed to be simple, supportive and intuitive.

One of the most valued features is the ability to process complex inferences efficiently, reducing the prerequisites required of the user. The user does not need to know the structure or nature of the data to visualize it, as the system is expected to extract the data in the most appropriate format for visualization and suggest further browsing and data summarization, thus delegating the entire process to the application. This user-application interaction could be called "inferential conversational access to information".

It is essential to note that the level of control given to the user must be extremely high. The user can ask for extractions that were never previously designed and can configure the data visualization, but also navigate and investigate the information without the need for an expert's intermediation.

In conclusion, the principal objective of this proposal, based on these elicitated results, is to create a product that enables complete control over data access and visualization through natural language and/or voice conversations while minimizing the need for user clicks or taps. The system must feature an optimized schema design, precise data visualization, textual data presentation, and a history management system that facilitates navigation across different entities. Finally, leveraging these capabilities and following some of the emerging desires, the proposal tries to include a potential use case for integrating the chat component into a more commercial-oriented module.

All these elicitated requirements are deeply investigated in the following sections.

3.1.1. Easy and agnostic schema annotation

The initial functional requirement for the platform is to establish connections with various relational databases, whether local or remote, managing common connection methodologies. Following this request, the prototype presented in this Thesis is primarily designed to manage relational sources, however, a connector for associative open data - i.e. data structures that store data in a key-value pair format - has also been included.

Users expect the application to extract relations, entities, and attributes directly from the DBMS metadata. As the interviews revealed, the product should also be resilient and implement inferences to reduce the steps and actions required of users. To make the platform user-friendly, the user only needs to provide some essential information about the nomenclature of entities and a few attributes. Indeed, the remaining activities are delegated to the platform's inferential capacity. This schema abstraction is necessary for the chatbot's navigation and helps to simplify the translation process from text to query through pruning.

The design phase must be easy and visually supported: it is crucial to represent the schema of various sources using a graph, making careful choices for massive database schemas, and allowing users to configure everything by selecting the entities of interest and providing annotations for each of them.

3.1.2. Powerful and efficient text-to-query translation

> *Show me all the grades of students enrolled in 2022*

The second crucial requirement of the platform is related to the generation and execution of queries based on natural language prompts. The system must identify the core topics of the request and use them to construct a working query using the knowledge extracted from the design phase.

To achieve this requirement, different text-to-SQL engines are available, and the current implementation utilizes OpenAI's GPT-3.5, which can be fine-

tuned for similar tasks, such as prompt-based queries or code commenting: one of its best-known derivated products for development support is GitHub Copilot [25].

To ensure precision, the system must be syntactically and logically correct. This feature becomes increasingly challenging as the query grows larger. In this Thesis, resilience to syntax errors and typos has been considered a non-functional requirement.

For these reasons, query generation must be able to handle literals with sufficient confidence level, for example using `full-text` or `like` where clauses. Moreover, the system must be able to self-correct in case of errors, thus increasing its dependability. Additionally, the system must prevent SQL injections and malicious query generation by carefully filtering executed commands.

Finally, the system must be efficient, returning results in acceptable response times, even when querying large operative databases. To this extent, a schema pruning step must be included to process queries with greater efficiency.

3.1.3. Precise and useful data presentation

The scope of this work extends beyond merely extracting data given natural language requests: indeed, the presentation of the extracted information is also of great importance.

As interviews confirmed, when processing operational data, it is necessary to present the data in understandable formats that can be further processed by humans to extract information and knowledge. Indeed, although tabular formats are useful for detailed investigation and are typical of relational extractions, they are difficult to interpret and should be replaced by precognitive visualization methods [26].

The system must reorganize the data to highlight the most relevant attributes: this step can be partially determined by the structure of the entities, but ultimately requires computation based on the nature of the data. These relevant attributes must then be represented through easily understandable and familiar graphs, as the success of data visualization depends on its ability to quickly transmit a message without requiring significant interpretation effort from the user.

The system must also be integrable with virtual assistants - e.g. applying text-to-speech (TTS) to natural language summary of tables - thus summarizing individual extractions from a high-level perspective and diving into the details of individual records as requested.

Finally, data mining processes are crucial for transforming data into information, knowledge, and insights. Summarization and mining are required to extract descriptive and predictive details and present them contextually with the user's requests.

Thus, we have reason to believe that this process, along with data visualization and data mining, is a key requirement for the data agents of the next future.

3.1.4. User manipulation of results

> *Count the grades per student through a bar chart*

The visualization of results cannot be an end in itself but must be part of a broader process of data navigation and access. This means giving the user the ability to manipulate the presentation to adapt the results to their preferred visualization. Navigation is fundamental for proper conversational access to information, which should not be considered a static element, but a malleable block.

To this extent, the chatbot must be able to correctly manage intents aimed at reworking the visualization of graphs and tables. The user should be able to reorder dimensions, filter, group and investigate displayed data. This relevant requirement is linked to modifying the data visualization - e.g. *"Show it using a bar chart"* - but also any further navigation and exploration.

In conclusion, the user must be able to move freely through the extracted results and also use these results to "explore" related entities. Manipulating the results is not just a matter of rearranging data visualization, but it is an entire suite of mechanisms capable of providing complete freedom to navigate and recombine the presentation of information.

3.1.5. Wiseful conversation management

Context coherence is crucial to ensure effective communication and understanding between individuals engaged in a dynamic exchange of information. Indeed, maintaining context across multiple requests is essential when using a conversational approach as an interface between users and data.

Although a conversational approach can be highly effective in bridging users and data, it is imperative to establish a system that retains a history of previous requests and leverages that information to handle future inquiries effectively. In other words, it is paramount to manage the history of messages, which can be used intelligently to satisfy future prompts.

Furthermore, it can be necessary to guarantee that the results of any requests can be persistently stored and re-accessed by the user. This requires storing the query results along with the commands used to obtain them. This feature is important because the system is required to update the results if the underlying data have changed over time, especially for dashboard interfaces where refreshing is common. By ensuring that results and commands are persistently stored, the platform can provide users with access to up-to-date and fresh information.

3.1.6. Automatized exportation of results

> *Create a dashboard about students*

> *Export this dashboard as a slideshow*

In the business context, it is not uncommon for data and information to be organized into summary panels, such as dashboards, and then further summarized into downloadable reports or presentations. However, dashboard generation requires a total change of approach, focusing on overcoming the technical difficulties and obstacles that inhibit users from freely generating and structuring their exports.

In this scenario, the concept of a conversation becomes more fluid and extemporaneous, evolving into a facilitated way of writing commands. Notwithstanding, the conversational approach to structuring individual indicators still remains crucial and therefore must be incorporated into dashboards.

In other words, conversational data environments should give users the ability to request entire dashboards through proposals and then configure each indicator through dialogues.

Furthermore, interviews have highlighted the importance of being able to export results: users should not waste time configuring each report but should rely on an automated export system. Therefore, it is the system's responsibility to *imagine*, generate and download reports on request.

Moreover, generating reports and dashboard exports does not mean simply saving indicators in PDFs, but completely rethinking the visualization. Indeed, static reports such as PDFs require different methodologies for presenting data. A simple example of this problem is the absence of result pagination in PDFs or slides, implying a new way of visualizing large datasets. As it is easy to understand, creating a slide presentation requires visualization methods that are very different from a PDF export. Timing, modes, texts, and graphics need to be reconstructed. Great importance, in this case, is given to serendipity: users should be able to completely rely on presentation proposals and extract inspiration for subsequent modifications.

4 | Approach

The process of defining our objectives played a pivotal role in the development of an infrastructure capable of meeting the escalating demand for data and processed information. By comprehending the functional and non-functional requirements, we were able to construct a modular platform that could flexibly accommodate various data sources, prioritize the back-end logic, maintain a streamlined design, and automate information access and processing. These modeling choices were guided by our objectives and requirements, yet they necessitated a careful balance with respect to their compatibility with the underlying architectures of the platform and their responsiveness to rapid evolution.

4.1. Declaration of intents

The primary objective of this proposal is to establish a reusable framework for creating, managing, and organizing lite suites that can extract and process structured data through conversation. The foundation of this development lies in **agnosticism** towards data sources, **simplicity** of interaction, **inferential delegation** to the platform, **modularity**, and **reusability**.

Indeed, the proposed approach prioritizes the definition of processes and pipelines as an abstraction layer, regardless of the technologies employed. The rapid pace of innovation in this field necessitates a modular and interchangeable infrastructure not tied up to a technology-specific approach.

Therefore, the focus is on defining the necessary and sufficient processes, pipelines, and functionalities to fulfill the information needs of our key users - middle managers in large corporations. Each component of the proposed infrastructure is interchangeable with equivalent modules, respecting established interface limits, thereby ensuring modularity and extensibility through adapters.

However, to develop a prototype, some technologies have been chosen, selecting them among the main technologies available at the time of writing.

In this section, we will examine the framework of the proposed platform and its internal processes.

The architecture is primarily composed of three fundamental modules:

1. Designer: The Designer module is at the core of the connection between the data sources and the application, and it involves defining the semantics of objects.
2. Chat: The Chat module is the central module for querying and conversational access to data, and it is closely linked to the Designer module.
3. Dashboard: The Dashboard module is an extension of the Chat module, which provides an interactive and expandable dashboard for accessing data.

Since the Designer and Chat modules are both essential and interconnected, they could be considered as a single and interdependent unit. The Dashboard module takes advantage of the previous modules' architecture to provide another way of accessing data that is widely used in the professional world: dashboards and panels. Therefore, it will be presented separately as an optional add-on to the platform, highlighting the framework's modularity and extensibility.

4.2. User journey

Following our commitment to an agnostic and user-centric approach, we have dedicated our efforts to create a seamless and straightforward user experience. As previously mentioned, our proposed solution aims to assist non-technical users who lack knowledge of the schema and querying techniques, as well as visualization methods.

To accommodate this assumption, we have developed an extremely simplified, user-friendly, and intuitive approach that extends to the configuration and dashboarding phases: the end user does not necessarily require technical or specialized knowledge of the underlying system. Additionally, the designer and end user's roles can overlap, allowing the same user to have full autonomy in both the configuration phase and subsequent activities.

Before utilizing the system, the user needs to identify and connect their own data source for querying. This source can be either local or remote. Once the connection configurations are provided, such as the database name, address, and access credentials, the user can start annotating entities.

The annotation process is straightforward yet crucial. For each table of interest, the user needs to insert an annotation in the form of a word or phrase that describes the table's content. The same applies to the columns within each relevant table, where the user declares their respective attributes. This process takes only a few minutes and focuses solely on tables and columns. The user is not required to declare links and associations explicitly. The annotation process is simple and only necessitates an initial user collaboration to assist the system in making subsequent inferences.

While not strictly mandatory, the user can provide general information about the entire database. For instance, they may specify that

> *Each table contains an ID field*

Once the desired database is annotated, the user saves their changes, and the system encodes the information, preparing it in the required formats for subsequent processes. These actions constitute what we refer to as the **design process**.

Upon completing this initial procedure, users can immediately initiate a conversation with the chatbot, with the only limitation being their imagination. After selecting the relevant database (which can include multiple sources), the user can enter their requests in the chat interface. In response, they will receive tables, texts, images, and graphs. Optionally, the user can dictate their requests verbally, which will be converted using speech-to-text technology.

Some responses may also contain clickable buttons that enable the user to take shortcuts, such as navigating between different topics or sections. These activities form part of the **inquiry process**, which, despite its simplicity from the user's perspective, represents the most complex pipeline from an inferential standpoint.

Furthermore, we have provided the user with the ability to independently create and save entire dashboards. To achieve this, the user selects their desired database and requests dashboard proposals based on a specific topic.

They can then use a textbox to type and configure the order, size, and title of various indicators, as well as add or remove them at will, in a *buttonless*, *codeless* and *touchless* manner.

Each indicator within the dashboard is essentially a separate chat session where the user can interact with the bot instance, requesting specific visual or content changes until the desired dashboard is achieved. Finally, our solution offers the option to export the dashboard as a file, such as a slideshow. The user can request this export via a text command, eliminating the need for a traditional UI, and it completes the **dashboard process**.

These three processes - design, inquiry, and dashboard - represent the three main activities of the system from the user's perspective (Figure 4.1). Despite being a complex and coordinated dance of automated tasks and user interactions, as described in the following sections, our solution ensures a seamless and user-friendly experience.

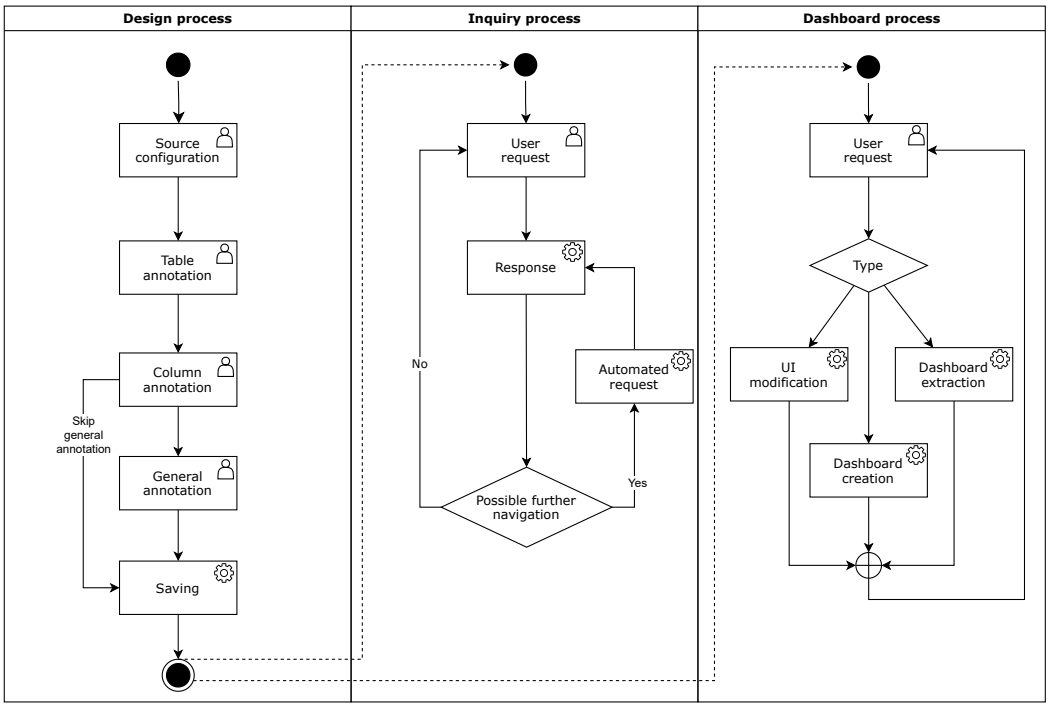


Figure 4.1: The three main processes from the user perspective

4.3. General structure

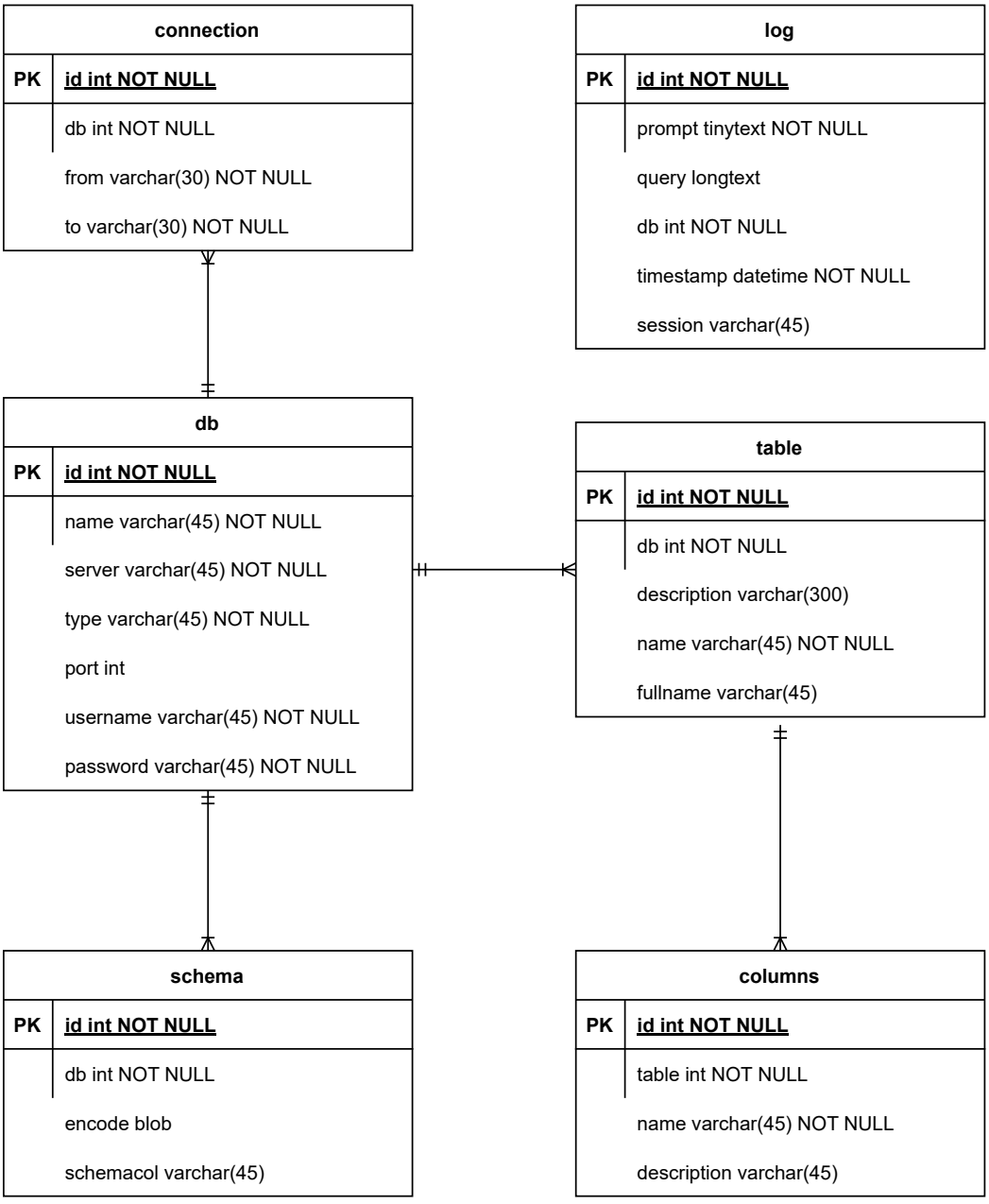


Figure 4.2: Metadata support database used by the system to collect schema annotation

Our software architecture app is built upon the classic 3-tier paradigm, comprising a modular back-end, a semi-modular front-end, and a support database. The app allows to create and configure a variety of connections in runtime, forming a network of data sources that users can query.

In this paradigm, connections can be directed towards both locally and remotely hosted data sources, making it not limited to just simple use cases. The great advantage of this structure is that each new connection can be easily added and the access to additional data sources is instantaneous and can be done in runtime, without requiring any kind of recompilation or specific training of the entire platform.

The app's database is designed to support all the metadata and abstractions for the user-defined connections and schemas.

The back-end is coded using a combination of two different environments:

1. An engine that manages APIs and API calls, making it a flexible tool for interfacing with various front-end paradigms (e.g., mobile, web, desktop APIs).

2. An ensemble of modules that interface with data sources, extract and process data, and transform it into information.

This dual-engine setup enables the back-end to efficiently manage complex data processing tasks and interface with different front-end technologies, providing a scalable and adaptable solution for a wide range of software applications.

The first component of our back-end is highly modular and easily extendable to meet the needs of different implementations, allowing for a flexible and customizable back-end that can be tailored to specific project requirements.

The second component is a modular environment that is called to perform data extraction, processing, and transformation tasks. Following this line of reasoning, each subcomponent of this module can be modified or removed, and new elements can be added to expand the functionality of the app, thus meeting the modularity and extendibility technical requirements.

Collectively, these elements combine to create a robust and flexible system capable of managing intricate data processing operations and seamlessly integrating with diverse front-end technologies.

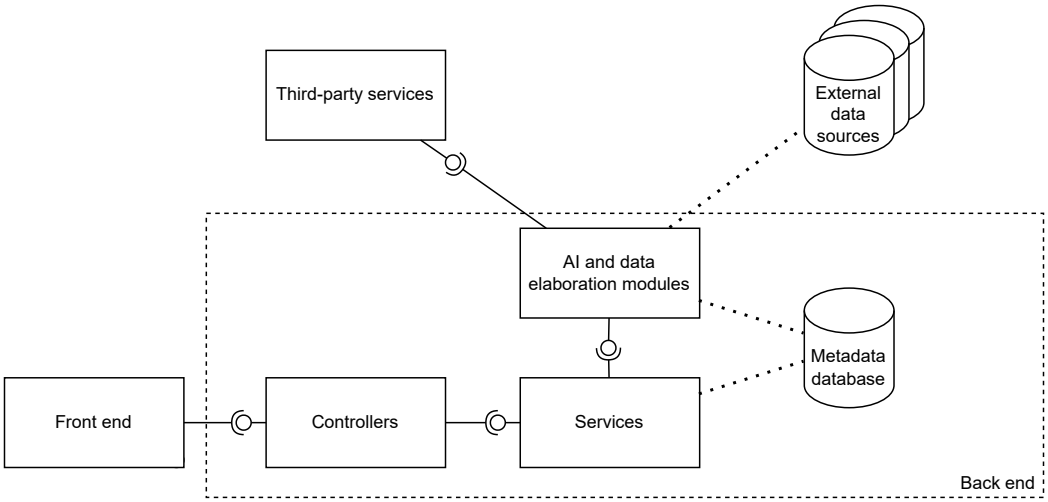


Figure 4.3: Back-end model

The explosion in popularity of pre-trained models such as the GPT family has required a rethink of how text-to-query processes are handled. GPT has positioned itself as a new baseline and disruptive technology. For this reason, it was decided to incorporate this paradigm into the platform rather than compete directly.

However, it is possible to replace this model with any other equivalent prompt-based tool or other NQL models as long as they are suitably predisposed to the interfaces proposed here.

For these reasons, the entire infrastructure uses GPT LLM both to transform prompts into queries and enhance our NLP procedures.

Therefore, the proposed platform organizes the metadata and schema annotations of the requested data sources, uses this information to drive a *few-shot fine-tuned* [9] GPT model to generate the queries, executes them, processes the results, and generates a text version of the results and graphical versions of data visualization.

Finally, the back-end orchestrates the interaction between multiple interchangeable modules of data summarization, data visualization and entity navigation by managing the conversation and historicizing requests. Moreover, it manages possible add-ons, such as the automatic generation of interactive data-based dashboards, slides, presentations and reports.

4.4. Design process

The **design process** of the meta-schema is a crucial task of our pipeline. Indeed, in order to operate, the framework requires knowledge about the semantics of the entities. Thus, the design process plays a crucial role in the platform. As mentioned earlier, considerable effort has been devoted to expediting, modifying, simplifying, and optimizing conventional ways of annotating and designing data sources. This agnosticism makes the platform more accessible to novice users who lack expertise in relational infrastructures and NLU requirements, which makes it challenging to interface with conventional schema annotation and tagging methods.

To address this, the chosen approach has been to minimize the user effort needed to activate the framework's operations. Furthermore, the scope of optional design activities that users can perform has also been reduced. The platform can produce results even without specific annotations due to the high level of inference of the text-to-query engines and the entire platform.

The platform has the capability to establish connections to various relational data sources, utilizing a combination of local and remote connections via SSH to access the databases. As a result, users are required to input just some details for each source they want to connect to.

4.4.1. Tables annotation

The design process starts with the selection of the data source that requires annotation. The designer component is built on the premise that the chosen database is modeled correctly, with entities, attributes, and their connections correctly defined. These logical definitions and links between entities must correspond to the actual schema of the queried database. If this assumption is respected, the interface accesses the metadata tables provided by the DBMS, thereby reconstructing the schema of the requested data source.

Users can view the schema in its entirety, with connections between entities highlighted, or, for wide databases, analyze the schema in a segmented manner.

Once the schema has been opened, the user will have to select the tables of interest and enter a series of attributes that can describe the use and

meaning of the table. Optionally, the user can also define a title for the table. This phase is extremely important: everything that will not be noted will be invisible during the interrogation phase and, therefore, cannot be visited. For this reason, it is important to note not only the main tables but also composite entities such as bridge tables, which are extremely useful for logically linking different entities.

Finally, the designer can add some general notes about the data source which may result helpful for text-to-SQL translations. An example could be:

> *In this database, the identity column is always called 'Id' for each table*

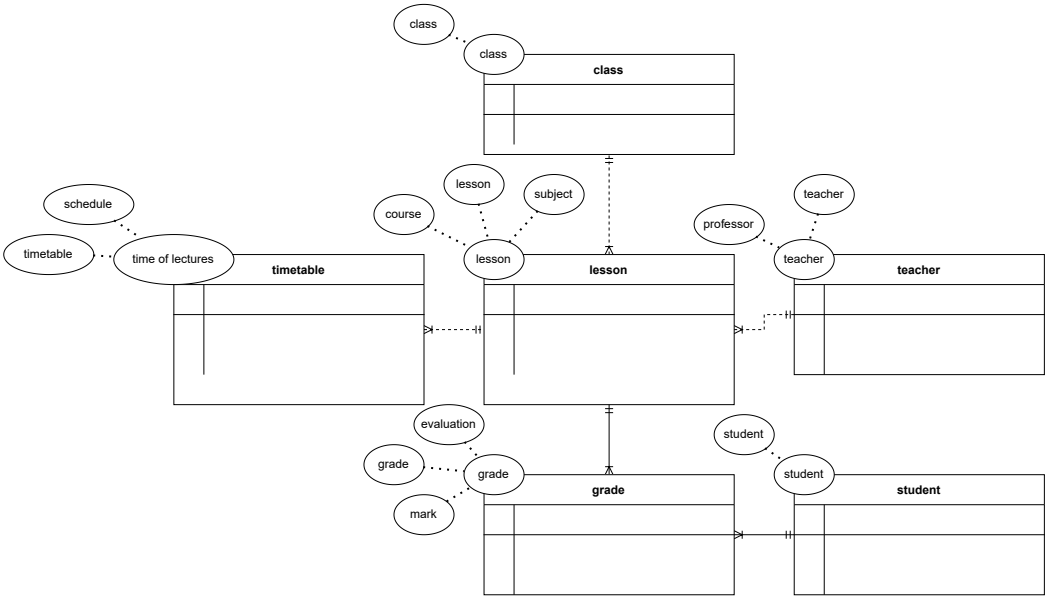


Figure 4.4: An example of schema tagging

In order to better understand the annotation methods, it should be remembered that a relational schema is not only characterized by a multitude of primary tables, but also by numerous support tables - such as bridge tables - capable of transferring many-to-many relationships in the actual schema of a database.

To facilitate access to complex data, it was decided to include not only the materialized tables of a schema but also any views. This choice allows not having to rely entirely on the interpretative power of the text-to-query model in deducing connections and links between tables, but also being able to count on persistent extractions developed by expert technicians.

In fact, it should be remembered that, although performing, given the small amount of knowledge required to operate on a database, the NQL model will not be able to offer appreciable results in all conditions.

To optimize table annotation, the user should prefer to use specific keywords that he/she believes will be used to refer to specific entities, thus avoiding using phrases as an annotation. The annotation of the tables is a mandatory step that provides the platform with the minimum knowledge necessary and sufficient to operate. However, as mentioned, other information may be included to increase the effectiveness of future extractions.

4.4.2. Columns annotation

In addition to the annotations on the tables, the user can decide to provide specific annotations on the columns to clarify the meaning of the attributes. This operation is not mandatory: if the columns are not annotated, the platform will use all the available attributes trying to make inferences on them. If this operation occurs instead, the text-to-query model will consider only the annotated columns, ignoring the rest.

4.4.3. Schema embedding

Once the annotation is completed, the user will have to save the changes and start the training phase. Unlike what happens in other systems, the training is runtime, i.e. it is managed directly by the platform as a function and does not require any accessory skills from the user. The embedding phase is performed by the platform as an autonomous sub-process without interrupting or shutting down the application itself. This means that it is possible to start embedding multiple data sources simultaneously on the same application.

The training phase is crucial for consolidating the annotations and is based on a process of feature extraction from the annotations. The chosen approach is, therefore, to transform each annotation into a feature vector through a word embedding model called SentenceTransformer. This model allows you to convert sentences and words into feature vectors based on the semantics of the words.

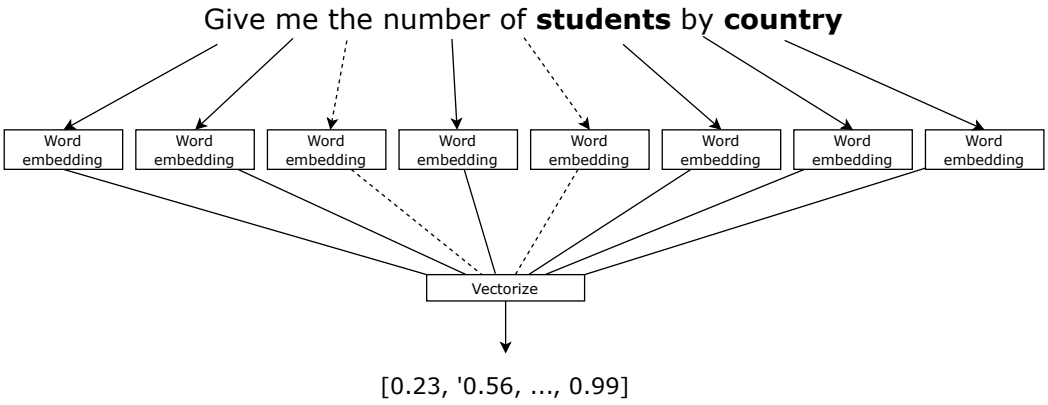


Figure 4.5: An example of word-wise embedding

For example, the words "tree" and "shrub" will be characterized by two different but similar feature vectors given the semantic proximity of the two terms. This would not be true between the words "tree" and "cup". In this case, the similarity can be expressed mathematically as inversely proportional to the cosine distance between the two feature vectors considered: the more two terms are semantically similar, the smaller the cosine distance is. Before passing the annotations to the encoding phase, it was decided to expand every single annotation through the synonymic search: this step allows to strengthen of the semantic search in the inquiry process. When the transformation process is complete, the schema will look like this:

```
{
"<table name>": [...<feature vector>]
}
```

This encoded representation is then saved persistently by the back-end. This step facilitates and accelerates the semantic pruning of the schema, which is fundamental in the following inquiry process. At each modification of the schema annotations, it will be necessary to save the tagging and proceed with a new encoding.

To summarize, the process of design of a data source follows these steps:

1. The first step is to connect to a database, which is mandatory.
2. Next, users can name the entities of interest and provide a series of attributes that describe their content. Rather than complete sentences, a series of tags will suffice.

   (a) Optionally, users can describe individual columns and exclude others. If no columns are tagged, the framework will consider all attributes relevant and autonomously manage incomplete schema descriptions. Possible developments include auto-learning of relevant columns.

3. Next there is the training: the annotations provided by the user are encoded through a fine-tuned seq2seq model and then saved. This facilitates semantic mapping and speeds up the pruning phase. The result is a persistent encoded schema that can be used to optimize the performance of the app.

Image preprocessing

One subprocess that is particularly noteworthy is the image preprocessing stage. The approach is designed to demonstrate the platform's flexibility by allowing users to include multimedia content in their search queries, primarily images. If the images exist as Binary Large Objects (BLOBs) in the database, users will be able to apply filters based on their content. To this extent, tables containing BLOB columns will be materialized. Then, if these BLOB attributes refer to images, each multimedia content will be converted into a textual description or caption as part of a preprocessing step. A pre-trained model is employed to perform this extraction task; once completed, the system persistently saves the records of these tables to be ready and accessible when needed.

These textual descriptions will be used to conduct semantic searches based on similarity with user prompts in the future. When a user requests an extraction on these tables, image descriptions will be included in the search clauses, if possible.

Finally, this approach can be appropriate because the tables containing images are generally static - e.g. product catalogs - thus characterized by low data frequency which makes them easily materializable without incurring data obsolescence.

4.5. Inquiry process

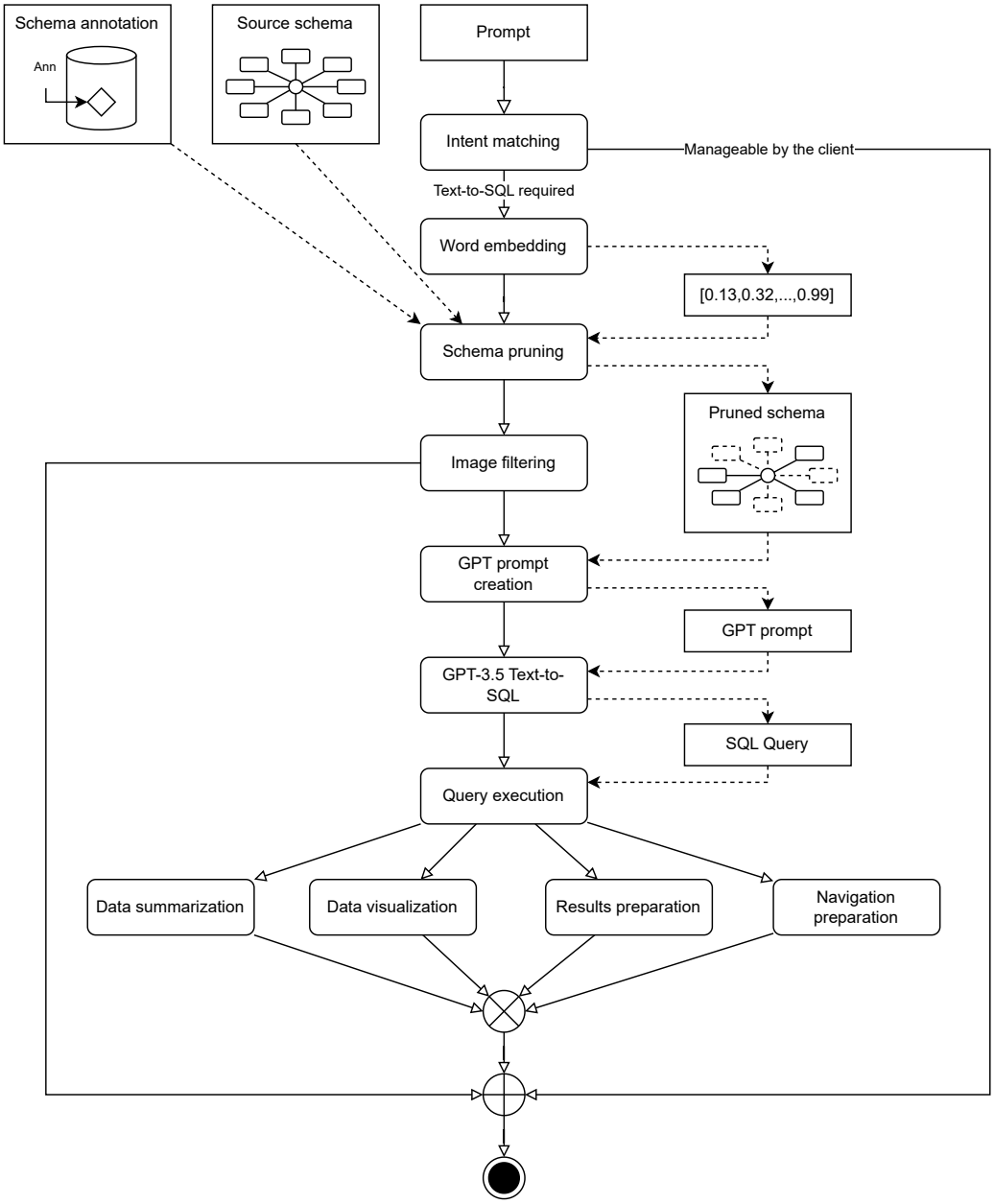


Figure 4.6: The inquiry process

The **inquiry process** represents the main interaction between users and data in the proposed solution. The inquiry is an extremely complex procedure that finds its fulcrum in the prompt-to-code model of GPT-3.5-turbo, however, requiring important computations upstream and downstream of the text-query translation. Also, when we refer to the inquiry process, we must bear in

mind that not all user requests are aimed at obtaining an extraction, but some interactions may refer to more specific commands such as reassortment of previous results, drill down or navigation to other entities.

To clarify better, the platform can manage three different query intents directly through the chat - an additional five in the dashboard component - which can be summarized in Table 4.1.

Intent	Example	Purpose
Select query	*Give me all the students*	New extraction
Navigation	*Get the grades of the student named "Francesco"*	Navigate to other entities
Data visualization	*Count the students per country using a bar chart*	Rearranging the results

Table 4.1: Inquiry intents

Each of these intents is handled with a NLU cascading fallback process: unlike what happens for the dashboard component, in the chat, an intent can be matched only if the previous handler failed to satisfy the request. This approach allows us to serialize and prioritize intents at the expense of temporal performance. In the proposed solution, the first intent to be tested is the **data visualization**, followed by **navigate** and finally **select query**.

For greater clarity, the three intents will be detailed in the following paragraphs, starting from the most important one related to data extraction. However, other intents are present and documented in the Dashboard process (Section 4.6).

The intent for a new extraction is certainly central and allows the user to access the data in a source given the metadata provided in the design phase. Indeed, to initiate an extraction from one of the connected sources, the user must have accurately configured the schema of the source in the design process. The inquiry process is triggered from the front-end, particularly from the chat page.

Following the cascade intent-matching, a new query intent is distinguished by a significant deviation from the previous requests, as it is captured after the fallbacks of the prior intents, indicating a dialog discontinuity.

It is challenging to define a precise archetype for this intent as it relies heavily on the user's phrasing and the complexity of the request itself.

Some examples of new query intent can be:

> *I would like to have all students whose name begins with F*

> *Which professor holds the subject with the best grade point average?*

> *Are there any overlaps in this week's timetable?*

As can be seen, the sentence structure is very changeable and this plays against an approach based entirely on intent identification.

4.5.1. Schema pruning

The first action, once such a request has been received, is called schema pruning, i.e. ellipsis of entities useless to satisfy the request.

The reasons for this activity are different:

1. the first, of a fundamentally technical nature, derives from the need to refine the search on specific entities by eliminating any noise that would make this agnostic approach inefficient when applied to massive databases.
2. the second, of a purely economic nature, derives from the use of the pay-per-use APIs: the wider a prompt is, the higher the fee for this request will be.

Therefore, reducing the span schema to work on allows you to minimize costs and focus only on relevant entities. To this end, we proceed with a semantic analysis of the user prompt to extract the main keywords of the request. This procedure takes place by exploiting a keyword extraction process capable of generating the list of main syntactic entities within the prompt.

Once this is done, feature vectors are generated for each of the keywords extracted by means of the same word embedding mechanism introduced in the design process. This allows, given a prompt, to obtain a list of numerical vectors corresponding to the most relevant elements and objects within the user phrase.

At this point, the encoded schema obtained in the design phase is used to associate a similarity value between the prompt and each table of the schema, then propagating this similarity also to adjacent entities.

Therefore, for each table and for each prompt, proceed as follows:

1. extract keywords from the user prompt using single-document unsupervised keyword extraction
2. given the encoded version of the annotations for each table, it is possible to calculate the table-prompt semantic similarity. The final similarity is computed by balancing two similarities.

$$\omega_i = \frac{sim_1 + 4sim_2}{5} \tag{4.1}$$

where:

sim_1: is the semantic similarity between the entire prompt sentence and the table annotations

sim_2: is the semantic similarity between the extracted prompt keywords and the table annotations

3. given the connections (foreign keys or bridge tables), the table-prompt similarity of each entity is propagated to the connected entities via a breath-in exploration of the graph (schema topology). The weights are propagated with progressive damping for each node (table) update given the following formula:

$$\omega_i = \sum_{j \in Ni} \omega_j \cdot d_{ij}^{0.08} \tag{4.2}$$

where:

ω_i: is the semantic similarity of the node i (from the previous step)

d_{ij}: is the minimum path length (number of foreign key navigation) between node i and j

N_i: are the neighbours of i defined as the tables connected with i through a foreign key $i \to j$

Through point 2 it is possible to obtain a first set of the possible entities that can be involved by the request.

However, this is not enough to decide which schema span is useful for fulfilling the request. Indeed, thanks to step 3, it is possible to include also adjacent tables in the initially selected span. This procedure is fundamental if we consider the relevance of bridge tables and the fact that they may not be included if considering only the prompt-table similarity.

Indeed, take this example:

> *Give me the courses attended by the student "Francesco"*

given the following schema:

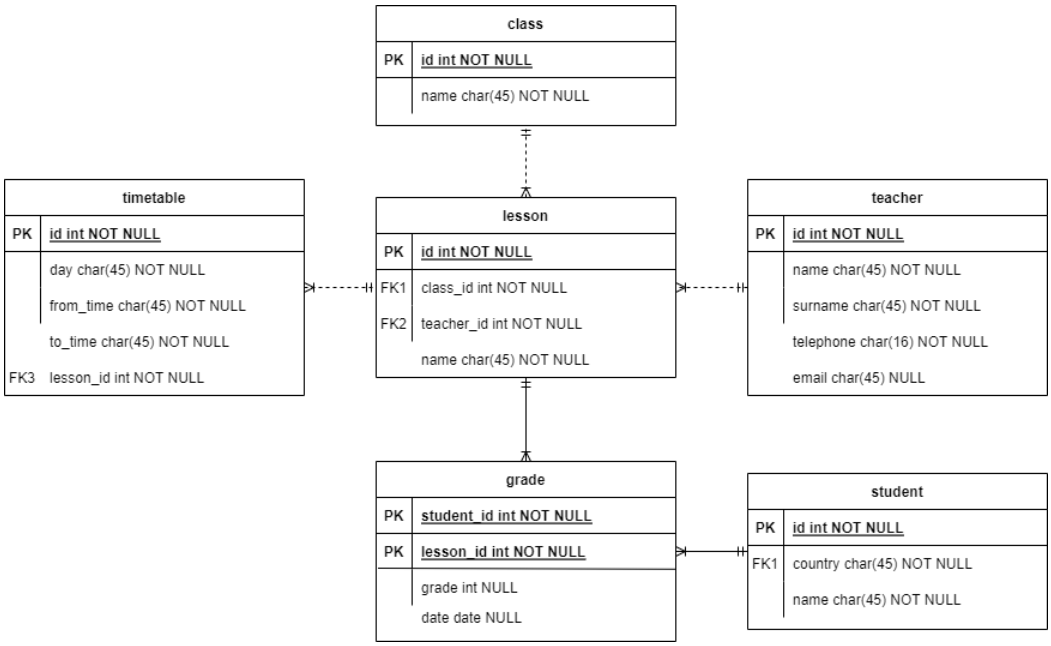


Figure 4.7: The considered database schema

The keyword extraction given the prompt is the following:

```
["courses", "student", "attended"]
```

Applying the semantic similarity step (Formula 4.1), we obtain this prompt-table similarity:

```
{"lesson": 0.4829, "student": 0.8703, "timetable":
   0, "class": 0.4293, "teacher": 0, "grade": 0}
```

Thus, through pure semantic analysis, the chosen entities would be *student* and *lesson*.

However, the many-to-many relationship subsisting between the two chosen entities is codified through the bridge table *grade* (containing also data about attendance) which must therefore necessarily be included.

To do this, the similarity weight of *student* and *lesson* must be propagated to all neighboring entities (Formula 4.2). By doing so, the *grade* entity will also be added.

```
{"lesson": 0.6903, "student": 0.8812, "timetable":
   0.1520, "class": 0.5788, "teacher": 0.2311, "
   grade": 0.6947}
```

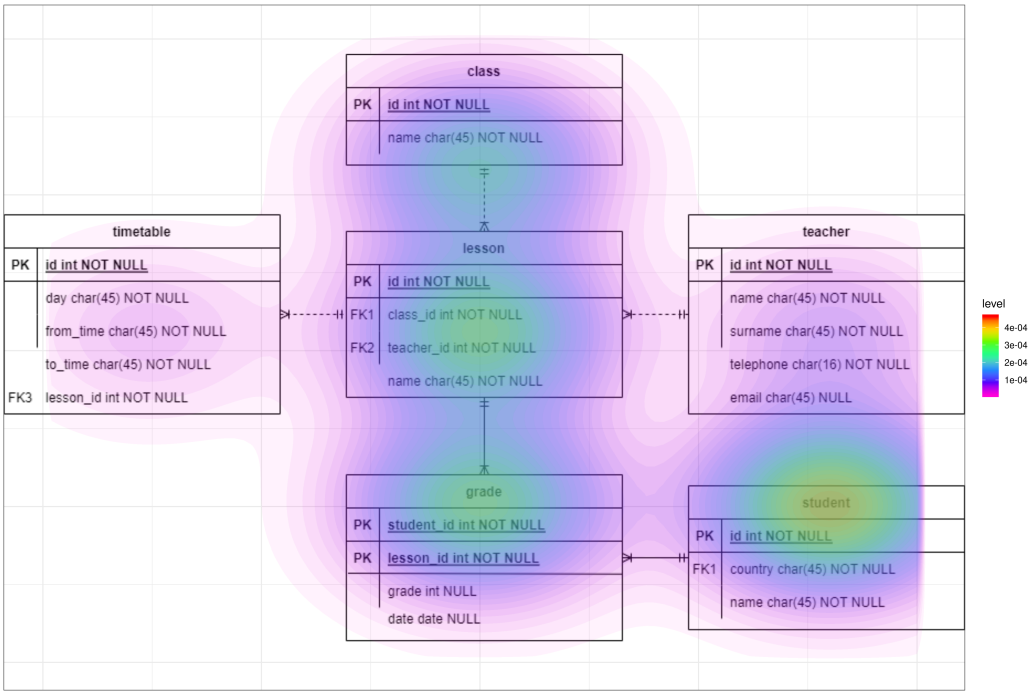


Figure 4.8: Prompt-table similarity heatmap

Once a mapping of all the entities with their relative similarity has been obtained, the model proceeds by selecting only a span of the entities that have prompt-similarity greater than 75% of the maximum value of similarity obtained.

```
threshold (75% of maximum similarity) = 0.6609
selected relevant tables = [lesson, student, grade]
```

The output thus obtained can be processed for the prompt generation phase. The overall procedure is formalized through Algorithm 4.1.

Algorithm 4.1 Schema pruning

Prompt keyword extraction
$encoding_1 \leftarrow$ Full prompt encoding
$encoding_2 \leftarrow$ Keyword encoding
for all table in schema **do**
    $sim_1 \leftarrow$ 1 - distance.cosine(attribute,$encoding_1$)
    $sim_2 \leftarrow$ 1 - distance.cosine(attribute,$encoding_2$)
    $\omega_{table} \leftarrow \frac{sim_1+4sim_2}{5}$
end for
queue $\leftarrow$ schema
while queue $\neq \varnothing$ **do**
    table $\leftarrow$ queue.dequeue()
    $\omega_{table} = \sum_{table \in Ni} \omega_j \cdot d_{ij}^{0.08}$
end while
$threshold \leftarrow 0.75 \cdot max\{\omega\}$
return $\underset{table}{argmax}\{\omega[\omega_i > threshold]\}$

4.5.2. Image filtering

After eliciting the requirements, it became clear that the conversational system needed to be able to access unstructured and multimedia elements using natural language. This would allow users to make inquiries related to content such as images, video, and audio, such as

> *Find me a teacher whose profile picture has a lake in the background.*

If the selected entity - the teacher in this example - is accompanied by an image in BLOB format, the system should perform the search by accessing the image content and applying a filter based on it.

To achieve this, the system materializes tables with BLOB columns during the embedding phase of the Design process and elaborates the images to extract a brief textual description (Section 4.4.3). While these materialized results may not be up-to-date, they are generally static data, such as product catalogs, making this choice justifiable. Later on, during the inquiry process, before creating an SQL query, the generation module checks for the existence of such materialization.

If found, it is used as a knowledge base for a subsequent semantic search on the textual description of the image.

To obtain the best result, prompts and descriptions are encoded via word embedding with `spacy` NLP, and the prompt-description similarity is used to rank the different results. If the similarity is greater than 0.75 (an arbitrary threshold), the record is matched and returned as a result; otherwise, the system proceeds with the generation of the GPT prompt and the creation of the SQL query, as described in the next section.

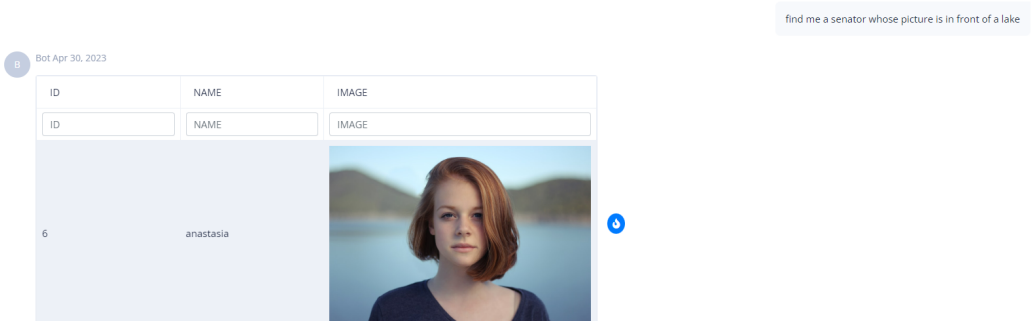


Figure 4.9: Example of image search[1]

4.5.3. Prompt generation and query generation

Query generation is entrusted to a GPT model based on prompts. GPT-3.5, like its successors, is a pre-trained model that can be refined through prompt optimization thus obtaining a fine-tuned model, in this case, aimed at text-to-SQL conversion. This methodology is called *fine-tuning few shots*[27] and takes place through the generation of a prompt including requests and specifications for the model.

In this work, we used few-shot fine-tuning [9] which is a technique used to adapt a pre-trained machine-learning model to a new task using only a small number of training examples. This is in contrast to traditional fine-tuning, which typically requires a large amount of task-specific training data.

It is crucial to emphasize that the query generation process relies solely on schema metadata, without utilizing any information about the data contained in the source.

[1]Photo by Christopher Campbell on Unsplash

As per the guidelines of OpenAI [28] regarding the use of instruction-based LLM, the prompt for the model has been developed by using XML tags to format commands. This method is used to prevent any possible prompt injection, which could lead to the generation of harmful SQL queries.

In order to further reduce the likelihood of such injections, the prompt has been designed to incorporate conditional prompting. This not only helps in enhancing the model's ability to generate accurate and useful responses but also makes it possible for the same prompt to support high-quality chitchat powered by GPT-3.5.

To increase the strength of the translation, the designer can insert general notes about the database. These notes can be extremely helpful to generate accurate translations fitted for the specific data source.

The resulting prompt is well-structured, efficient and capable of producing reliable output, while also enabling the language model to engage in casual conversations in a natural and engaging manner. Finally, to increase the accuracy of SQL generation, the prompt includes a few examples of text-to-SQL translations, following the literature.

In the context of NLP, few-shot fine-tuning can be used to adapt pre-trained language models, such as GPT-3.5, to perform new tasks. This is achieved by providing the model with a small number of examples or demonstrations of the new task, along with a natural language prompt that describes the task. The model then uses this information to learn how to perform the new incoming activities.

For example, if we want a model that responds like a wise old man, we could use a prompt like this:

```
# Answer me as if I were a wise old man, using a
   rich vocabulary and many aphorisms and anecdotes.
# Reply according to this format: [OldMan:<reply>]
#
# User: How are you?
```

Given this characteristic of the GPT model used, we proceeded to define a prompt template that optimizes results, is flexible to adapt, and reduces exploration expenses.

The template is organized into three sections: the first one defines the response's features, the second one displays the last processed queries, and the third one exhibits the schema span obtained from previous pruning.

The schema is linearized and includes annotations to aid the model to generate queries based on object semantics, trying to maintain a compact yet complete format.

Additionally, foreign keys are included in the linearized form.

```
# Use LIKE and description or name columns.
# Schema is complete and no other column exists.
# Use alias for COUNT, MAX, SUM, MIN, AVG
# General information: <general information>
#
#
# [mysql|mssql|oracle] schema:
# <schema>
# foreign keys:
# <fks>
# last query: <last query>
#
# prompt: <prompt>
#
# SELECT
```

Where `<schema>` is codified as:

`<table name>(<annotations>): <column1>|<column2>|...|`

each per line (\n) while foreign keys (`<fks>`) are linearized following this format:

`<table name>.<col name> = <ref table name>.<ref col name>`

GPT has a high-speed query processing capability, but the accuracy of its output is heavily reliant on both the end user input and the design of the prompt template, which can be adjusted as needed.

SQL Injection and efficiency

The output generated cannot be directly executed, and the fine-tuned model needs to undergo a filtering and processing phase. In order to prevent SQL injection attacks, we exclude any commands that are not compliant with the relational algebra principles of extraction and projection.

Additionally, to reduce the response time between the request and the display, we implemented pagination, cutting the maximum results count to 15 records per page. This is also motivated by the existence of resource-intensive queries. Indeed, due to a lack of direct knowledge about the entities' cardinality, some SQL scripts may be extremely requiring and lead to performance issues.

Error management

Another important point is related to error management: solidity and resilience to errors are critical success factors for this type of service. Given the information source agnosticism and the high level of inference required to generate such results, some queries may suffer from typos, non-existent fields, or worse, logical errors.

Error handling takes place according to three different cascading methods:

1. Using heuristics it is possible to correct typos and field names by searching for such columns in the list of existing attributes. This allows the system to diagnose but also fix some errors, subsequently proceeding to execute the correct query.

2. If the error cannot be solved heuristically, the error code is forwarded to GPT attaching it to the query, which will attempt to correct it. This approach works well for relational algebra errors but tends to fail for logical errors.

3. If the error persists, the prompt is re-sent for a new run of GPT. Of course, this can also happen at the request of the user.

Through this downstream control, it is possible to ensure high self-healing by increasing the dependability of the entire platform. Another important factor of prompt-based models is the possibility of maintaining a historical context and processing it. This functionality is highly used in this platform both to interpret user messages that can be linked to previous extractions

and to self-correct errors deriving from the model itself. A deeper analysis of error management is reported in Section 4.5.7.

4.5.4. Navigation

Navigation allows users to move through the schema and extract information without needing a global view of the entire database. Navigation can be initiated through direct prompts or by utilizing hints generated with each extraction, which are displayed as button chips alongside the search results.

For instance, returning to the previous example, connections between *lesson* and *class* could potentially connect the user to the entire schema graph. These suggestions are made possible by an inferential system based solely on the infrastructure of foreign key constraints. During the design phase, the user is not required to declare any connections as they are automatically deduced from the logical structure of the database.

To achieve this, the module considers the tables used for the extraction and search, along with the columns present, and determines possible foreign keys that can be exploited to navigate through the database. To accomplish this, tables are extracted back from the query using Regular Expressions (RegEx) up to aliases, while foreign keys are extracted from the information schema tables.

4.5.5. Data summarization

In some cases, raw data may not lend itself well to visualizations, making a conversation-based approach confusing. Consider the use of a platform solely through vocal means, such as virtual assistants, where presenting results in a tabular or graphical format may be challenging to communicate effectively.

While it is possible to linearize tabular formats into sentences and present them verbally, this approach may not be innovative, summarizing, or effective in capturing the essence of the data. To address this issue, a data narration feature is introduced that leverages statistical heuristics and inference to express what data reveal.

To implement this feature, a small module is developed that also supports displaying data in a textual format. Algorithm 4.2 outlines the process: statistical analysis and attribute recognition are performed first, followed by

collecting information on distributions and tabular characteristics. Finally, natural language augmentation and paraphrasing are used to harmonize the result, also exploiting LLMs.

Algorithm 4.2 Data narrative

```
Attribute types extraction
stat ← Statistical analysis on distributions
relevantColumns ← Relevant columns based on distribution
for all column of relevantColumns do
    text ← text + description(column.data)
end for
text ← rephrase(text)
return text
```

4.5.6. Data visualization

Alongside the narration of data, the module also looks for potential graphical visualizations of the results. Graphic formats are considered essential to complete data storytelling, which is optimally realized through the dashboard component, where graphic and narrative formats are maximally fulfilled.

The platform is capable of producing various charts based entirely on a heuristic approach, depending on the relevance of the attributes. The decision was made to prioritize two-dimensional visualizations, as other methods may be more challenging to comprehend and more susceptible to inferential errors.

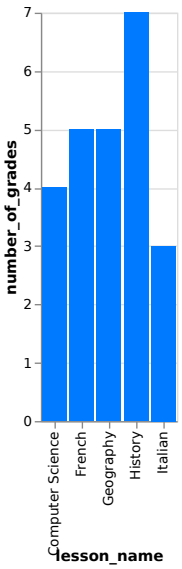
Thus, the focus was on implementing graphs such as bar charts, pie charts, line charts, heatmaps, and maps, relying therefore on pairs of dimensions.

The available views are summarized below, along with the respective conditions for which each view is appropriate. A unique feature of the module is its complete delegation of visualization to the front-end, provided it is capable of executing and interpreting JS. The module does not return static images or pure graphic schemes but instead delivers pure JS, which can be readily injected to generate interactive graphics.

Below all the graphs currently supported are listed, along with the data type combinations required for each chart type.

Bar chart

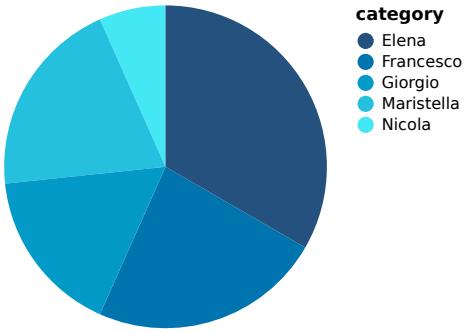
A bar chart is employed to compare categorical data and it comprises rectangular bars whose lengths are proportional to the corresponding values they represent. For example, a bar chart can show the sales of different products in a month or a year.



- Categorical data + numerical/temporal data
- Two numerical data
- Two categorical data

Pie chart

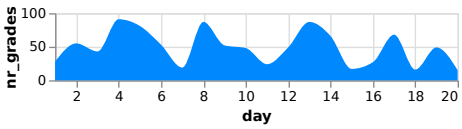
A pie chart is utilized to illustrate the relative proportions of various categories within a whole. It features a circular area divided into sectors, each representing the percentage corresponding to a specific category. For example, a pie chart can show the market share of different smartphone brands.



- Categorical data + numerical/temporal data
- Two numerical data

Line/Area chart

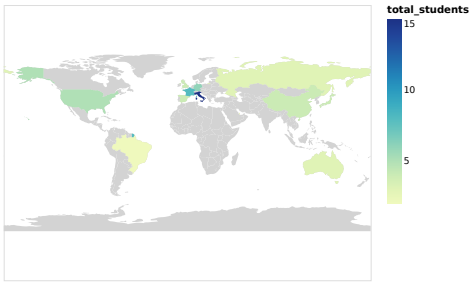
A line chart is used to show trends or changes in numeric data over time. It consists of a series of points connected by lines that represent the values of a variable at different points in time. For example, a line chart can show the stock price of a company over several years.



- Categorical data + numerical/temporal data
- Two numerical data

Map chart

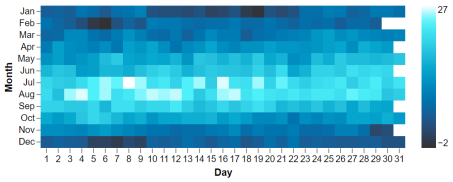
A map chart is used to show geographic data or to compare data across regions. It consists of a map with different areas shaded or colored according to the values they represent.
For example, a map chart can show the population density or the GDP per capita of different countries.



- Country data[2]+ numerical data

Heat map calendar

A heatmap calendar is used to show patterns or variations in data over time on a calendar. It consists of a grid of cells that represent days, weeks, months, or years, with colors indicating the values or frequencies of a variable on each date.
For example, a heatmap calendar can show the number of visitors or the revenue of a website on each day of a year.



- Temporal data + numerical data

In addition to being able to access the data, the system must allow the user to operate on these results by manipulating them according to his needs, for example changing their view. To better investigate this intent, it was decided to dwell on a few relevant requests that are very common in the data processing phase.

Indeed, many of the requests are about grouping, decomposing and swapping view sizes. To do this, the platform can recognize four main intents: grouping, pivoting, filtering and changing the type of chart. The visual catalog for such changes is reduced to the data visualization that accompanies the new extracts but manages to satisfy many of the more common demands.

[2]An attribute is considered to be geospatial-related if containing ISO codes, alpha-3 or complete country names and the system is completely autonomous in identifying this type of data

Here are some examples of user prompts that fall under this macro-intent:

> *Plot the results grouping by country*

> *Count the grades per student*

> *Show me a bar chart for these records*

As already mentioned, in this case, we do not proceed with a new extraction, but we work on the latest results obtained. To facilitate this process, the back-end temporarily saves the results of the last query per session, so that it is possible to alter these views by communicating the session GUID and the user prompt.

This module can recognize not only the intent but also the entities useful for these operations, whether they are parameters for filtering or columns for pivoting or grouping.

To maintain linearity and continuity between on-demand user visualization and automatic data visualization, the output of the process is identical, leveraging the front-end for the rendering burden for graphics. Finally, considering what has already been said, the fallback of this intent is interpreted as an entry point for "New query intent".

4.5.7. Query explanation and self-correction

From a first analysis, it emerged that there exist three possible macro outcomes for a user inquiry.

Executable and successful query The first case derives from a correct execution of the generated query which represents the construction of a syntactically and lexically executable query. Thus, the commands are correctly interpreted and produce at least one result - i.e. at least one record exists.

Notwithstanding, little can be said about the command's semantic-logical correctness which instead derives from the interpretative and understanding capabilities of the model. In this case, the results are transmitted to the front-end, displayed, summarized and presented to the user who will evaluate their logical consistency.

Erroneous query In the second opposite case, the query is syntactically or lexically incorrect. It can be caused by erroneous formation of the query structure, or by lexical errors or misalignment concerning the database schema or SQL dialect. The outcome is that no records can be presented to the user.

Thus, to increase the system's resilience, if the DBMS propagates a syntactic or lexical error, the model modifies the query up to a maximum of 2 successive occurrences. For example, if the `Unknown column "Name"` exception is thrown, the model will try to remove that column from the query, maximizing the probability of extracting at least one record.

Within the proposed pipeline, this micro procedure has been called **self-correction** and, unlike the other components, it does not make any inferences but forcibly corrects the queries according to some specific error codes. This operation can compromise the reliability of the results, but, at the same time, it gives the user the possibility to visualize an outcome and decide subsequent requests based on it.

Executable but ineffectual query The last case represents a hybrid condition of the two above. If the query were syntactically correct, the DBMS would not generate any errors. However, it is not uncommon that the generated commands do not produce results due to misinterpreted parameters or literals or logically incorrect query operations - even if executable. For example, a misinterpreted filter can lead to no results being found, while still generating an executable query.

On the user side, these last two cases give the same result on the front-end side: `No results found`.

It was immediately clear how this type of system's response was not verbose enough, thus, not very useful. While wanting to mask technicalities to the user, it was then decided to proceed with a **query explanation**.

This procedure takes the query as input and performs a natural language explanation and comment. To this extent, the model uses again a GPT model with a few-shot prompt that emulates the generation of code comments. This more comprehensible and explanatory result is passed to the front-end, displaying the command explanation and giving non-technical users the possibility of understanding what the model has tried to do.

For instance, comments include details about the parameters used and what the query attempted to do. Therefore, the user will have the tools to understand if the model has, for example, misinterpreted a literal parameter, or if it has the wrong entity for the query. Consequently, the following requests can be refined, indicating the correct steps to the model. Here is an example:

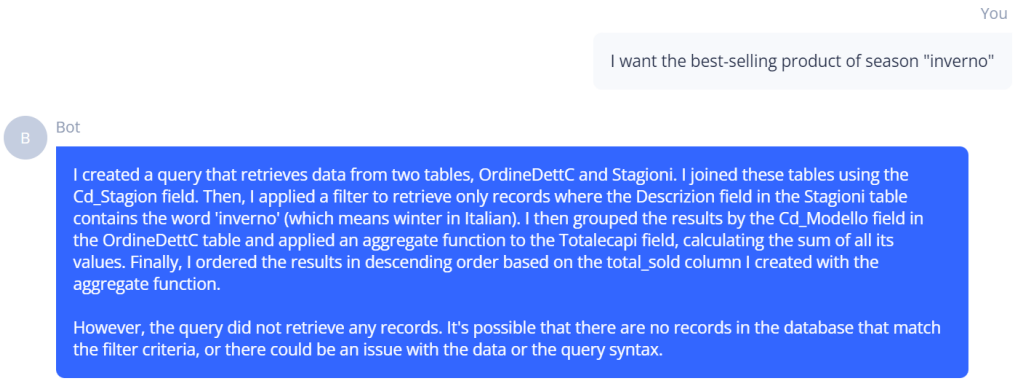


Figure 4.10: Example of query explanation

4.5.8. Summarized inquiry process

The inquiry process starts with the front-end sending a natural language request. The body of the request is composed by the prompt and the connection ID to the requested data source.

1. Upon receiving the request, the model evaluates whether the message refers to a previous data visualization (DV) request through a cascade check: if it is not possible to understand a DV intent, the process continues to the next step.

   Otherwise, in case a DV intent can be identified, the architecture accesses the temporary history of the last extraction and proceeds by mapping the request to infer and create the best type of chart, identifying the entities involved (columns/attributes) and the type of operation requested (grouping, counting, average, etc.).

2. The request is processed using embedding to identify the most relevant entities and topics in the request (e.g. *"Orders"* when the request refers to sales extraction). These topics are encoded, and their encodings are compared with the encoded schema. More specifically, for each entity in the encoded schema (encoded tables), the similarity is calculated with

the keywords extracted from the request. This likelihood is obtained by computing the cosine distance between the feature vectors of the encoded topic and the encoded table. Once this is done, each table has a similarity value with respect to the request.

However, it is also necessary to consider how different connections (FK) influence the need to include tables in the considered span: for example, given a table B as a bridge between A and C, if A and C are relevant to the user prompt, the table B must also be considered relevant. To do this, the database graph is traversed using the declared FK - a prerequisite is that the DB is correctly defined -, and the similarity weights of each table are propagated to nearby entities, dampening them the farther one gets from the considered table. A table-similarity dictionary is returned, which is used to order the most relevant tables and select the best ones.

3. Once the pruning is obtained, the prompt is created to be sent to a prompt-based LLM - e.g. GPT. Through an API call, a SQL Query is obtained. A query evaluation phase is carried out: the command is parsed to avoid harmful code, and direct searches are replaced with fuzzy searches. At this stage, images are considered if any.

4. The query is sent to the target data source in order to be executed. In case of an error, the request is re-assigned, returning to step 3. This allows for a first level of self-diagnosis and self-healing.

5. The results of the query, if available, is sent to the first component of the back-end, which completes the response with post-processing. Given the available fields extracted from the query and the topology of the DB, all the entities that can be visited are identified.

6. Post-processing the data involves data visualization (DV), data summarization (DS), or, in case of failed extraction, query explanation (QE).

   (a) DV is done by selecting the most relevant columns based on statistical factors and creating a chart. The result is a scheme that defines the main dimensions of the chart and its visualization.

   (b) DS is done by studying the statistical characteristics of the result table, identifying the most relevant dimensions, summarizing the overall content of the table, and narrating the columns' data.

(c) QE is done by rephrasing the query into a human-understandable text.

7. The results (raw records), the chart, the summarization, the linked entities and information on pruning are combined into a single object and forwarded to the front-end for visualization.

4.6. Dashboard process

While acknowledging a form of primacy of conversational access to data in single extractions, in many cases it is faced with the need to monitor different aspects and dimensions of the same topic. This truth is widely confirmed in the interviews carried out with a sample of middle managers whose approach to accessing information does not take place through single indicators, but thanks to interactive panels. In other words, one of the most common approaches is to monitor different parameters (KPI) simultaneously capable of disclosing more information on a given field of investigation, usually preferring visual or purely textual extractions rather than tabular results.

The dominance of visualizations through **dashboards** is evident in many working environments where the aggregation of information and relative communication takes place preferably through visual and interactive reporting.

For the reasons mentioned above, it was decided to broaden the perspective to include tools to support the generation of dashboards in an automated and easy way, even for less expert users. Furthermore, this realization must exploit the processes already consolidated in the solution to give relevance to the entire reporting process.

The construction of dashboards is a very complex procedure that must take into consideration multiple structural and cognitive aspects, with an important and deep understanding of the topics, entities and relationships on which the dashboards stand.

It is therefore undisputed that this intrinsic complexity is heightened by the objective of giving users full autonomy in the generation of information interfaces by exploiting natural language.

Imagine being able to directly request the development of a dashboard just by describing the goal of your export.

At that point it would be possible to request:

> *Generate a student dashboard at the university*

and then modify it simply by writing:

> *Add me a pie chart about students' provenience*

> *Remove the penultimate chart*

> *Increase the width of the first KPI*

Finally, to complete the interaction, the user may require:

> *Export this dashboard as slides*

and thus obtain a presentation in a completely automatic, conversational and data-based way, exploiting the semantics of the extractions, the inferences on the visualization modalities and NLP to generate a more human narrative. Having set the objectives of this process, the problem was tackled by dividing the procedure into 4 macro-activities which were then orchestrated at the application level.

1. NLU, interpreting user prompts and extracting core entities from user prompts. This operation is necessary and crucial for starting the subsequent processes.
2. Dashboard proposition given the topic proposed by the user. This activity requires the construction of some queries based on semantic inferences and the knowledge deduced from the data source.
3. Query execution and visualization are the final and conclusive steps of generating a dashboard. This activity is greatly facilitated by the reuse of the inquiry and chat components previously introduced, thanks to the low coupling imposed during the development phase.
4. Export and reporting are optional steps in the process that complete a user experience based totally on technical delegation and a conversational approach. This step requires an in-depth understanding of the topic and the best visualization method to present data in precognitive and usable formats.

4.6.1. NLU

The interpretation of requests remains the crucial step of the conversational approach. For this component, it was decided to entirely generate a natural language interpretation model.

This methodology makes it possible to exercise direct control over the definition and management of intent matching and entity extraction, both of which are essential for carrying out the actions requested by the user. The model can understand 5 different intents shown in Table 4.3

For each of these intents, it is possible to extract certain entities useful for the creation of the interface. The entities and their uses are shown in Table 4.2

Intent	Entity
Add/Remove/Modify	Card index
Create	Topic/Filters
Export	Format

Table 4.2: Dashboard intents and entities

Intent	Example	Purpose
Add	*Add a card about students average grades*	Enlarge dashboard indicators
Remove	*Remove the third indicator*	Reduce dashboard indicators
Modify	*Enlarge/Reduce the last card*	Rearranging the results
Create	*Create me a dashboard about grade*	Generate a dashboard proposal
Export	*Export the dashboard as a presentation*	Generate a downloadable reporting

Table 4.3: Dashboard intents and examples

The model runs in the background as a singleton service that can interact through continuous messaging. This mode of interaction and execution allows us to save resources by loading the model at the start of the back-end itself and increasing the temporal performance. Moreover, multi-user handling is achieved by exploding the computation into a pool of independent and autonomous threads.

The module, therefore, returns the response regarding the intent and the entities extracted for each request, forwarding them to the front-end which will take care of executing the connected action. In case of a match with the "Create" intent, the back-end takes charge of the request by transferring it to the following Dashboard generation module.

4.6.2. Dashboard proposal generation

Frequently, users face challenges when attempting to organize and structure complex and aggregated extractions. Defining elements and indicators that can be useful for decoding such complex phenomena is an exceedingly difficult task, requiring extensive knowledge of the entities and information essential for comprehension.

Consequently, this Thesis proposes a reusable method for creating dashboards by maximizing the use of the conversational interface. For these reasons, we have incorporated a module capable of suggesting indicators that can be useful for comprehending a phenomenon or a topic.

Thus, the system suggests various extractions related to the topic of interest, starting from its definition and displaying the results. This tool does not aim to replace human contribution, but it serves as an excellent source of ideas for future improvements and extractions. In summary, this process allows the user to produce reports investigating specific dimensions of the requested topic while simultaneously overcoming writer's block.

Unlike the single extract presented in Section 4.5, the generation of SQL queries in the context of dashboards is characterized by greater degrees of freedom. If in the chat component, the requests are specific, in the construction of a dashboard the scope is broad and ensures greater freedom for the platform in proposing and building indicators.

The approach chosen in this case is strongly guided by the study of the relevance of potentially interesting fields and entities given a fairly generic topic. The construction consists of the following steps:

1. Given the "topic" extracted by the NLU module, the potentially involved entities are highlighted by reusing the schema pruning process illustrated in 4.5.1. This implies a study of semantic similarity between topics and annotations of the schema introduced and elaborated in the Designer component, exploiting the encoded form of the schema. Downstream of the selection through pruning, the platform increases (or decreases) the topic-entity similarity score based on the number of user requests involving each specific entity. This passage, therefore, returns a list of eligible entities that constitute the center of the research.

2. For each of these entities, the relevance of the attributes is deduced by exploiting a statistical-semantic approach. The underlying assumption is that the stronger the column's relevance is, the higher its internal variance will be and the less it will be related to the identity columns of the entity. More formally, the relevance score for each attribute is calculated as:

$$\omega_i = \frac{|\theta_i|}{|A_i| * \sigma_i * 2} + \omega_{i,T} + \frac{\omega_{i,d}}{2} \tag{4.3}$$

   where:

   ω_i: the relevance of attribute i of table T

   $\theta_i = \{a_j\}_{j \in A_i}$, the set of unique values contained in attribute T_i

   σ_i: the correlation between attribute T_i and T's identity attributes

   $\omega_{i,T}$: the semantic similarity of T_i name and T table name

   $\omega_{i,d}$: the semantic similarity of T_i name with possible descriptive attribute names

3. Given the eligible entities and the relevant columns, the module proceeds with the construction of the queries starting from 18 templates capable of exploiting inner joins ($\bowtie$), grouping, statistical functions and sortings. To increase serendipity, the templates are modified (augmentation) and chosen randomly: this prevents the proposals from becoming excessively fossilized on the history of user requests.

This procedure takes place by selecting different columns and tables, testing the syntactic construction of each query for a maximum of 500 iterations or until the completion of 6 extractions.

Once the queries for the individual dashboard cards have been obtained, the back-end sends this information to the front-end which is responsible for displaying the results.

4.6.3. Query execution

The final step is to display the results. As already mentioned, we wanted to reuse the components already introduced previously, thus exploiting the developments regarding data visualization and data summarization. The methodology used exploits an emulation of user prompts originating from the front-end which requests the execution of a specific query from the back-end.

The processes that follow the execution of the query are instead shared with the inquiry process already presented and focus on the search for a representative format (data visualization) and a summary of the results (data summarization). Each query is executed in parallel and the results are displayed in a separate chat component: this allows the user to select each single proposed card and interact with the results according to the methods already exposed for the Chat component. This choice is aimed at demonstrating the reusability of the components and the inquiry processes, due to a design paradigm aimed at reducing the coupling of the components themselves.

Below, in Figure 4.11, each white card is a single chat thread that can be used to modify the visualization or the content for each indicator. Moreover, by using the input field above, users can configure the positions, sizes and titles of each card below.

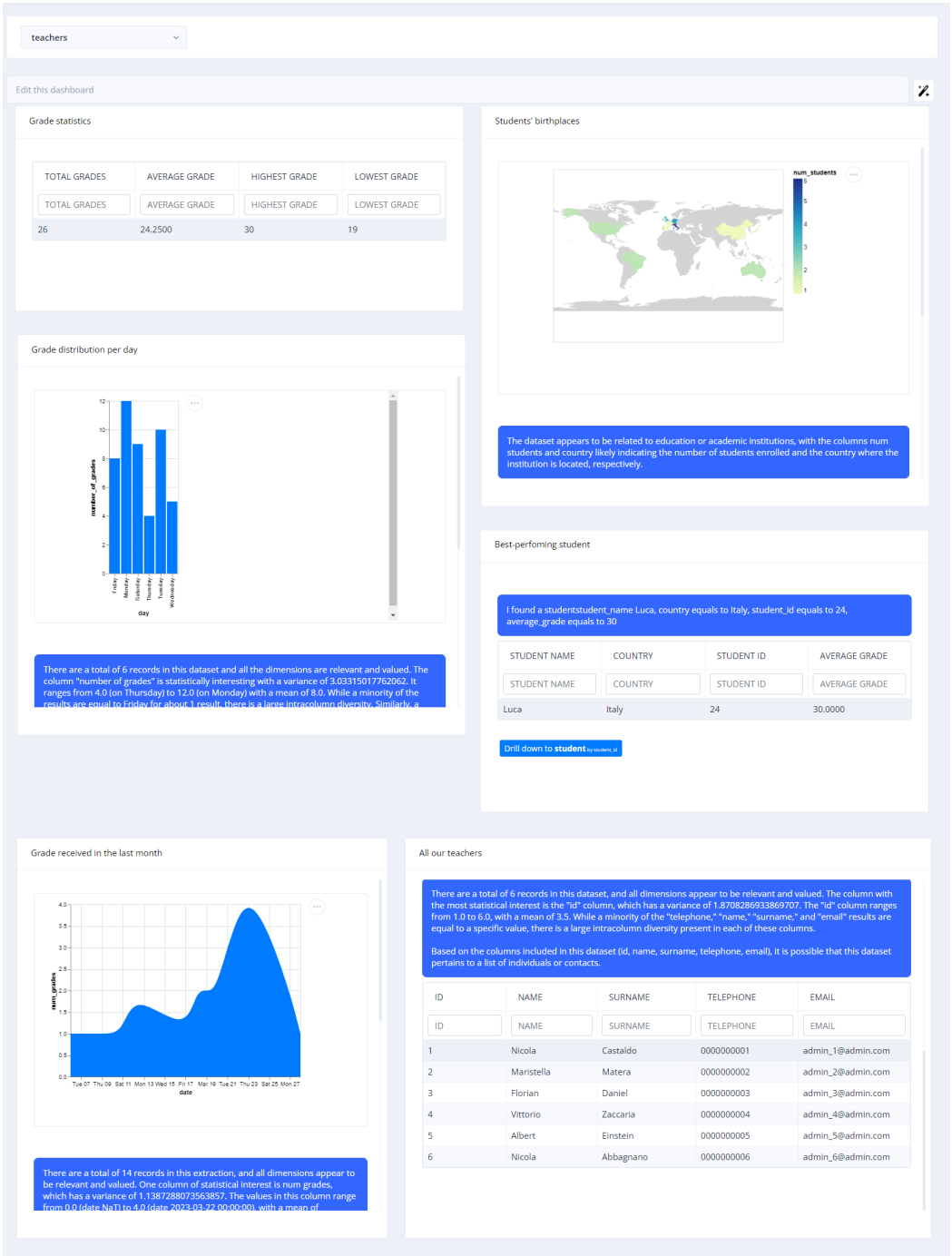


Figure 4.11: Autogenerated panel from "Create a dashboard about students situation" request on a school database

4.6.4. Export and reporting

After the chapter on accessing and representing information through dashboards, an innovative contribution has been introduced relating to the export of the results obtained.

Exporting information requires not only preparing it for a pre-established format but also understanding the audience and how to present the content effectively. The slide export is a perfect example of this module, which is strongly based on visual impact, message conciseness, and presentation minimalism, maximizing the precognitive effect on the reader. To achieve this, the system re-runs all queries to be exported and re-elaborates the visualization formats to favor a graphic representation over a tabular format for each extraction. Similar to the data visualization process expressed in Section 4.5.6, the module infers the best visualization methodology based on the type of data and attribute relevance.

Moreover, for this type of export, the narration of data is crucial, requiring the engine to generate a coherent and attractive narrative path of data storytelling. For this purpose, the module exploits a user-provided title for each indicator and combines this information with the query in order to obtain a complete description for each set of results.

Then, these descriptions are further refined: a paraphrasing fine-tuned GPT model has been introduced to make titles and texts more natural. The contribution of an NLP engine is essential in obtaining precise, syntactically correct titles and subtitles suitable for different contexts where this extraction tool is applied.

Finally, three templates - with six layout pages each - are used to structure the skeleton of the presentation and increase the reusability.


Analyzing Academic Performance: A Comprehensive Look at Student Grades, Birthplaces, and Teacher Impact
Discovering the Truth Behind Our Grade Statistics
About
26 Grades
Exploring the Diverse Birthplaces of Our Students
Just
2 students
Unveiling the Identity of Our Best-Performing Student
It's
Luca

Figure 4.12: Dashboard `pptx` export example

5 | System Architecture and Implementation

In this chapter, we will explore the implementation of our prototype conversational data access platform, called **Queric**, describing its architecture and the design choices we have made. At the same time, the tools we used to develop the system are illustrated.

5.1. Framework Architecture

The architecture of this framework (Figure 5.1) is designed as a 3-tier application, with the presentation, application and database tier. The presentation tier consists of several components, including the designer component, single chat component and dashboard component.

Each component exchanges data with the application tier, specifically with the controllers, which act as the intermediary between the presentation and the application level. The dashboard component reuses the chat module and allows users to view their data in a visually appealing way.

Moving onto the application level, the tier consists of various Python modules and NodeJS components. The Python modules are responsible for tasks, some of which are schema pruning and query execution. These Python modules are singleton and interchangeable, and they can exchange data with a NodeJS component called Python Adapter. To ensure efficiency, the Python modules use cached temporal results called Resources.

On the other hand, the NodeJS component consists of controllers, services and connectors towards the target data sources and the internal metadata database.

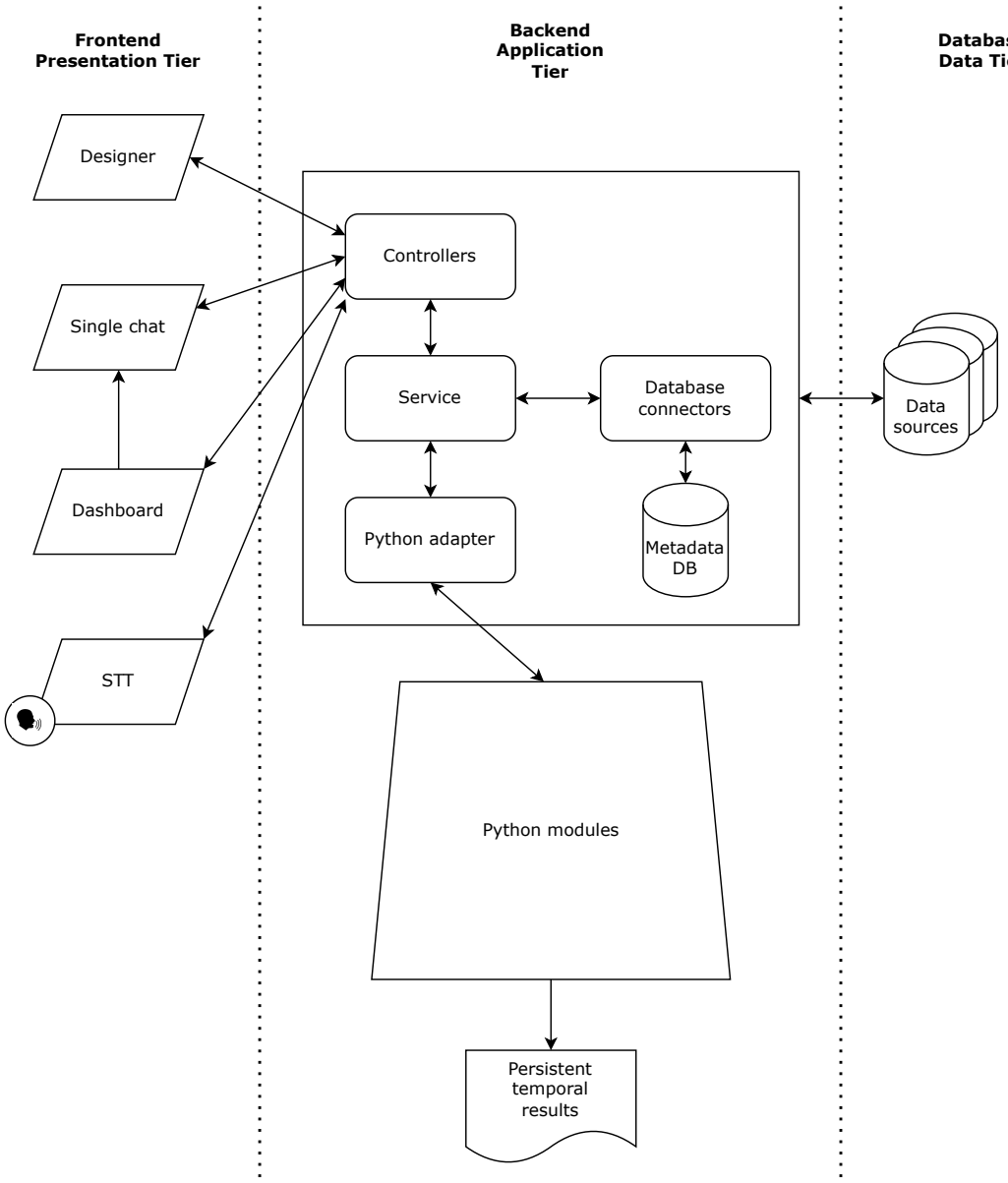


Figure 5.1: Framework architecture

The NodeJS controllers exchange data with the front-end and use services to implement business logic. The NodeJS services interact with Python Adapter to access Python modules and database connectors in order to reach external data sources but also to access metadata databases, which store annotations and metadata about sources.

The metadata database is an internal MySQL database that stores all the necessary information about data sources, such as data types, data ranges, and data annotations.

With this architecture, the framework is designed to be scalable and efficient, providing a seamless experience when accessing and analyzing their data.

The technology and design patterns selected for the conversational-data-access platform are an excellent fit for the project's goals. Using NodeJS, Angular, and MySQL creates a scalable, efficient, and reliable *triptych*. The back-end, developed using NodeJS, excels at handling large amounts of data and has a non-blocking event-driven architecture, therefore allowing for optimal performance.

Looking at the front-end, Angular provides a flexible and powerful framework for developing front-end components. Last but not least, MySQL, on the other hand, offers a robust and secure relational database system that can handle average volumes of data and enjoys widespread use and support.

Furthermore, the platform's 3-tier architecture has a clear separation of concerns among the presentation, application, and data tiers. This separation allows for independent development and management of each tier. The presentation tier comprises the designer component, single chat component, dashboard component, and a potential external chat interface. These components communicate with the application tier via the controllers. The application tier contains Python modules and Node.js components, with Python modules responsible for tasks such as schema pruning, query execution, data visualization, summarization, tagging, and dashboard proposal.

The controllers exchange data with the front-end and use services; these services interact with Python connectors to access Python modules but also with database connectors to access external data sources or access the internal metadata DB.

The design patterns used in this platform, such as the Singleton pattern and the Model-View-Controller (MVC) pattern, provide numerous benefits, such as increased modularity, maintainability, and scalability. The Singleton pattern ensures that each Python module is only instantiated once, improving memory usage and performance. Undoubtedly, the use of this pattern enables a clear separation of concerns between the application's data, presentation, and control logic, making it easier to develop and maintain each component independently.

Therefore, the technologies and design patterns used in this conversational-data-access platform provide a robust, scalable, and efficient platform for accessing and analyzing large volumes of data through natural language conversations.

The use of Node.js, Angular and MySQL, combined with a 3-level architecture and well-established design patterns, makes the platform highly adaptable, modular and easy to maintain: for these reasons, this union has been chosen for our implementation.

Finally, looking at the interaction between the front-end and the back-end, the client queries the application layer through REST API, whose structure is exposed through the back-end Swagger.

The back-end system generates a JSON file that includes text messages, results, the executed query command, schema pruning information, specifications for displaying charts, a summary of the results, and additional information that may be useful for the user. The interfaces between the Python Adapter and the modular ecosystem of functional components in Python are illustrated below and can be found in Appendix B.

5.1.1. Interfaces

As previously stated, the infrastructure is structured around a system of interchangeable components that are modular and connected by common interfaces. This design allows for easy reusability of the components as needed. The network of interfaces plays a pivotal role in facilitating communication between the front-end and back-end, as they expose REST APIs, thereby orchestrating the first two levels of the 3-tier architecture. These interfaces are documented in the appendices and exposed via Swagger.

In particular, the back-end leverages the interfaces exposed by the NodeJS connector to access the local MySQL metadata. The interfaces are also crucial in enabling interaction between the application back-end and Python modules. To manage these interfaces, the Python Adapter was implemented, whose primary function is to facilitate message exchange between different components while optimizing computation.

Python Adapter

The adapter is a crucial service in the back-end application, functioning as a singleton that initializes when the application starts up. Its primary task is to manage the interaction between NodeJS and various Python components. To achieve this, the adapter opens a server socket that is accessed by NodeJS acting as a client. This component is launched by the NodeJS back-end during the startup.

More precisely, a socket server adapter is a software component that allows different modules or applications to communicate with each other using sockets [29]. Sockets provide a way for two or more processes to exchange data over a network sharing dependencies and injections workloads. This method of software design has been included in a lightweight format in order to speed up the modules' imports as well as increase the modularity by splitting the infrastructure into C/S.

The NodeJS forwards requests through the call

```
python client.py <action> <argv>
```

which reaches the adapter at the socket port. `argv` represents the set of possible parameters specific to each module/action. From there, the adapter distributes the actions to the appropriate realizers, which implement the requested functions, constituting a hybrid facade pattern. Each action corresponds to a specific Python module that is injected into the singleton adapter during loading. As long as the interface between the adapter and the module is respected, these components can be freely replaced. The interfaces are documented in Figure 5.2 and in Appendix B. The singleton middleware service is preferred for its ability to optimize the response time and throughput of Python components. By sharing dependencies and processes among different modules, latencies are significantly reduced, especially in the case of loading massive Natural Language Datasets like `en_core_web_lg`.

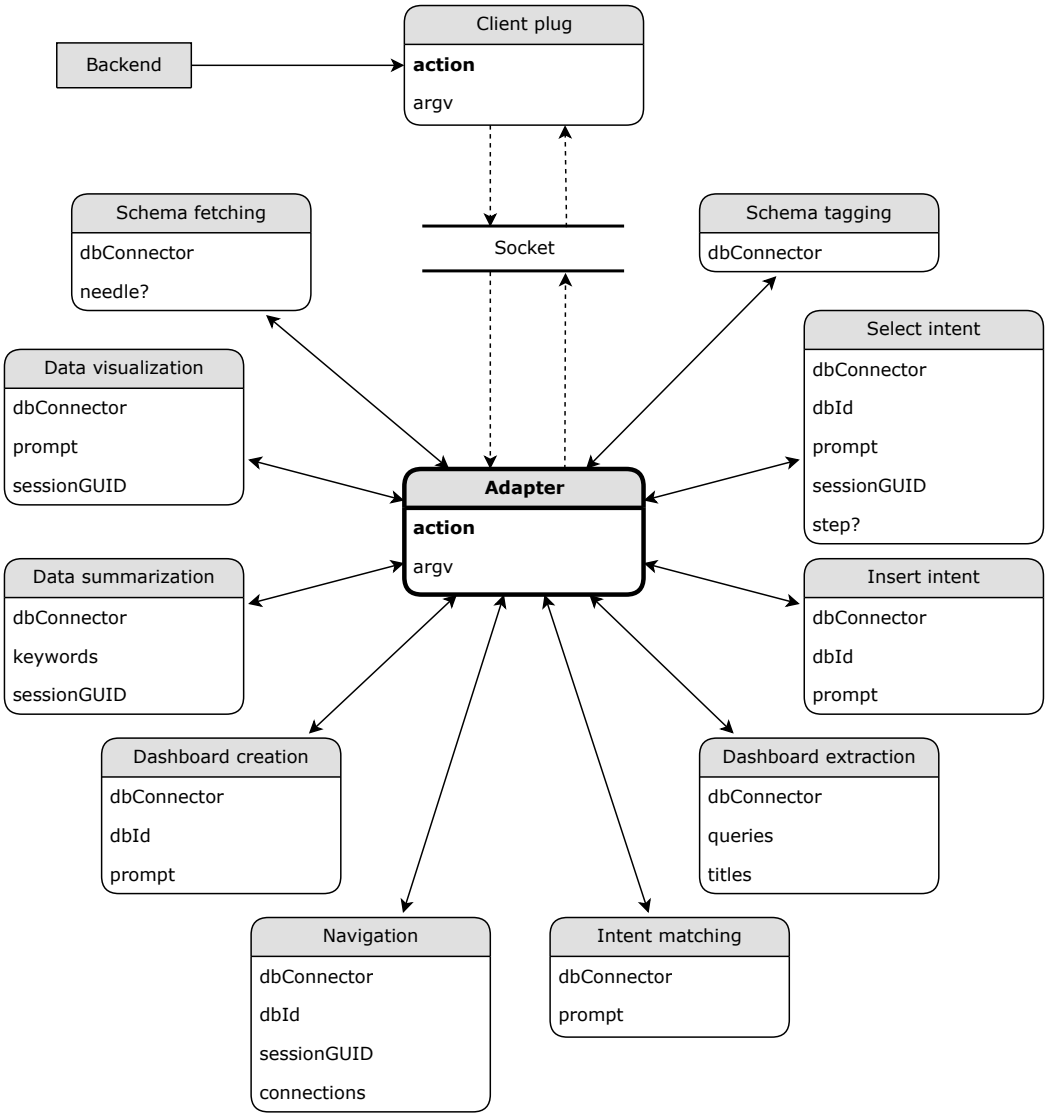


Figure 5.2: Adapter interfaces

5.2. Conversational infrastructure

Among the different modules introduced in the previous chapter, the most relevant one is represented by the conversational engine, thus this section focuses on the architectural decisions that were made to determine the internal structure of the chat and dashboard components.

To provide a clear description, we will refer to Figure 5.3, which illustrates the organization of each module within the system. The architecture can be divided into three macro components.

5.2.1. Conversation manager

The conversational management module handles the reception and transmission of messages from and to the end user. More precisely, there is a direct correspondence between the conversation manager and the front-end. It comprises two main components: the *connectors* component, which communicates with external chat interfaces, and the *visual component blocks* to render chat and dashboards. Internally, the chat component displays the conversation and comprises internal modules for rendering graphs and messages. In addition, the component has the ability to display specific button chips to facilitate navigation. The *chat component* is also reused entirely in the *dashboard component*, where it is combined with the management of card objects that compose the panel's layout. These elements use a set of NPM libraries to optimize the rendering process, such as the pivot table or the DB schema visualization components.

5.2.2. Application manager

The back-end application manager is the central module of the system, responsible for processing incoming messages, managing access to data sources, and presenting the results in a correct format. *Controllers* receive and handle API requests, relying on associated *Services* to handle response logic. These services, in turn, call upon the Python modules which manage the core logic of the system.

To maintain the conversational context, each chat session is marked by a unique session GUID, and session history is temporarily stored as a resource for later access during processing. Controllers also manage concurrency, creating new threads to handle requests until responses are sent.

The Python modules serve as the backbone of the infrastructure, enabling GPT to interface with databases and metadata extracted during the design phase. They include a *schema tagger* for schema design and annotation tasks, *schema pruning* for tailoring schema based on requests, query executors for creating and executing queries through database connectors, *Data visualizers* and *Data summarizers* for processing results, and a dashboard definer for NLU management activities and dashboard generation, all supported by various Python libraries.

The *dashboard generation* component is the only part that relies on the native RASA NLU engine for intent decomposition. The NLU module structure is not depicted in Figure 5.3, but follows a standard RASA infrastructure. Finally, the Utils unit is essential for managing the complexity and load of the entire back-end, offering simple helper methods to assist in these tasks.

5.2.3. Data sources

The entire infrastructure relies on several connectors to remotely query data sources but also to interact with the local database containing metadata.

The majority of the used connectors are based on *over-SSH* protocols. The database connectors over SSH enable secure and encrypted communication between the infrastructure and remote data sources. This is particularly useful when accessing sensitive data, such as financial or personal information, which requires strict security measures.

Similarly, the database connectors based on ODBC provide a standardized way of accessing data stored in different types of databases, regardless of the database's vendor or location. This allows the infrastructure to access data from multiple sources in a uniform way.

The *local database* containing metadata about each remote data source serves as a catalog for the distributed data system. It stores information about the remote data sources, such as their location, structure, and access credentials. This metadata allows the infrastructure to identify and connect with the remote data sources, as well as to query and retrieve data from them.

Moreover, we prioritize and guarantee the security of data by ensuring that **no information about the raw records from customer data sources is transferred or used outside our application**. This means that third-party services like GPT never have access to data themselves, except for schemas.

5.2.4. Resources

To improve system performance and maintain persistent annotations made during the design phase, various resources are utilized. For instance, the schema annotations for each data source are stored in a local database, and categorized as either table or column annotations.

To foster the performances of NLU processes, each *annotated schema* is encoded and locally saved in `.dictionary` files, accelerating the pruning phase. The information about the relevance of tables' attributes is stored persistently as well in `.relevance` files.

Session history data are also saved locally, but in a temporary format, with a user-configurable schedule for automatic cleaning. The files are saved in binary format, except for the session history, which is stored in JSON format.

5.3. Implementation

As previously mentioned, the platform's development is the result of the integration of various technologies and design patterns.

On the front-end, the main development relies on Angular 14 framework, which utilizes Typescript 4.7 along with HTML and SCSS. The infrastructure uses different NPM libraries to enhance development and improve the results. Unlike interpreted coded Web products, Angular's results must be compiled before publication. This technology allows the developer to distribute the product as a website or mobile Progressive Web App (PWA). Additionally, the front-end can be compiled and distributed as a library. For the infrastructural back-end, a NodeJS 19 server framework is used, developed with Typescript 4.7 and Javascript, enabling easy deployment of the server with the same front-end requirements. As with the front-end, the application layer uses different NPM libraries and can be distributed as a library.

The logic back-end comprises various `Python 3.10` scripts that realize interchangeable modules that can be replaced as needs or technologies evolve. To connect with the underlying data layer running on `MySQL 15.1` DBMS, these scripts use the `mysql-connector` library. For connecting with MSSQL sources, `pymssql` is used. To perform NLU-related tasks, `rasa_nlu`, `nlpaug`, `sentence_transformers`, `nl4dv`, and `nltk` are used to extract entities, match intents, and compute similarities among topics. To graph different charts, the application uses a combination of `vega-lite` and `altair`. For coding PowerPoint presentations, `python-pptx` is imported.

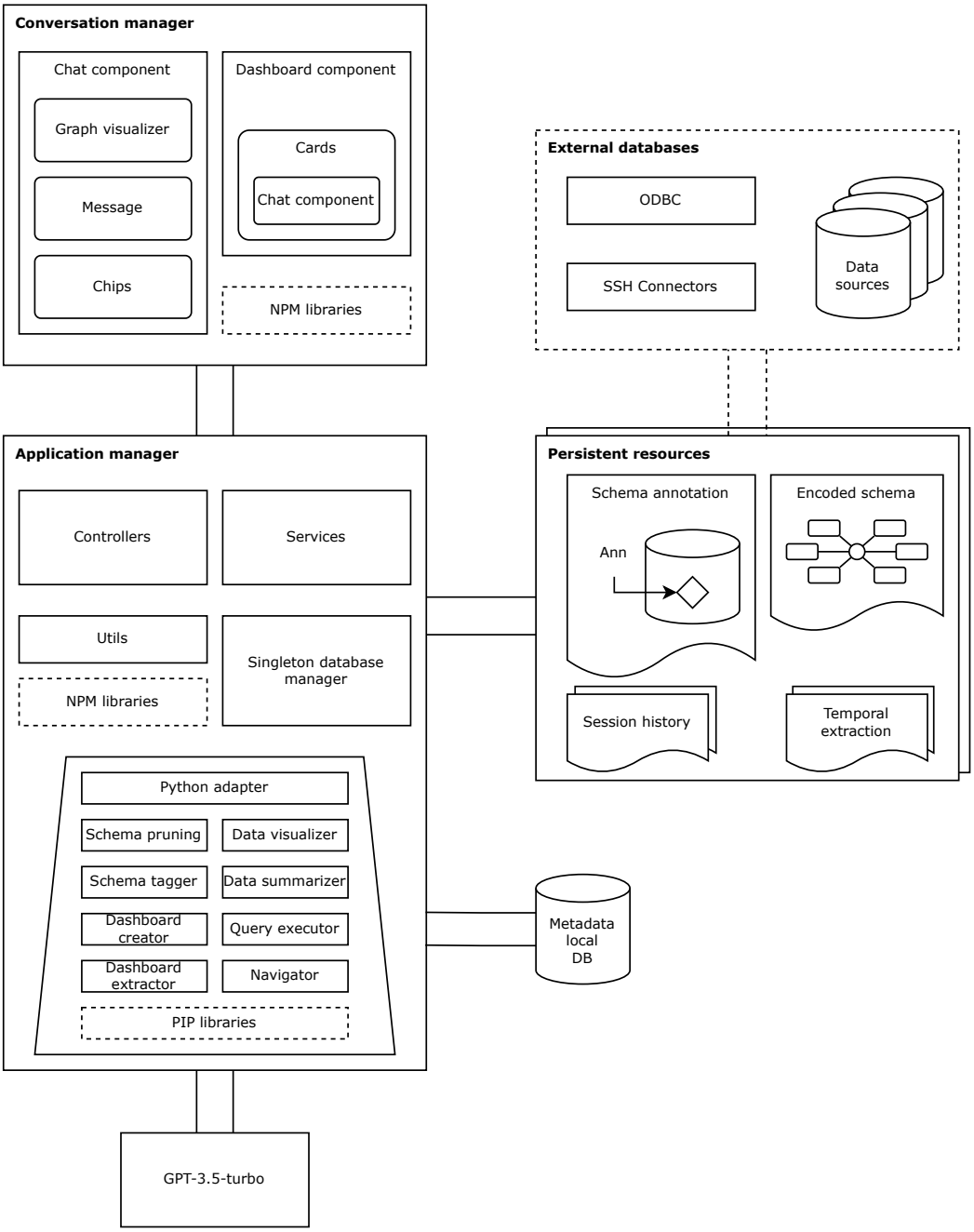


Figure 5.3: Proposed chat architecture

6 | Evaluation

This chapter extensively analyses the technical and qualitative results obtained through efficiency and effectiveness measures and user tests.

Unlike previous chapters, the entire experiment was conducted on a real, high-dimensional operational ERP database, provided by the SYS-DAT Group company[1], which comprised 251 tables having between 8 and 139 fields each. This context is considered an extreme yet common scenario in corporate environments, and the choice to test the platform in such conditions aimed to ensure the evaluation's realism. To conduct the tests, the Designer component was tagged, reducing the scope of interest to the 17 most relevant entities of the data source and a total of 196 columns. The schema is graphically shown in Figure 6.1, omitting the columns.

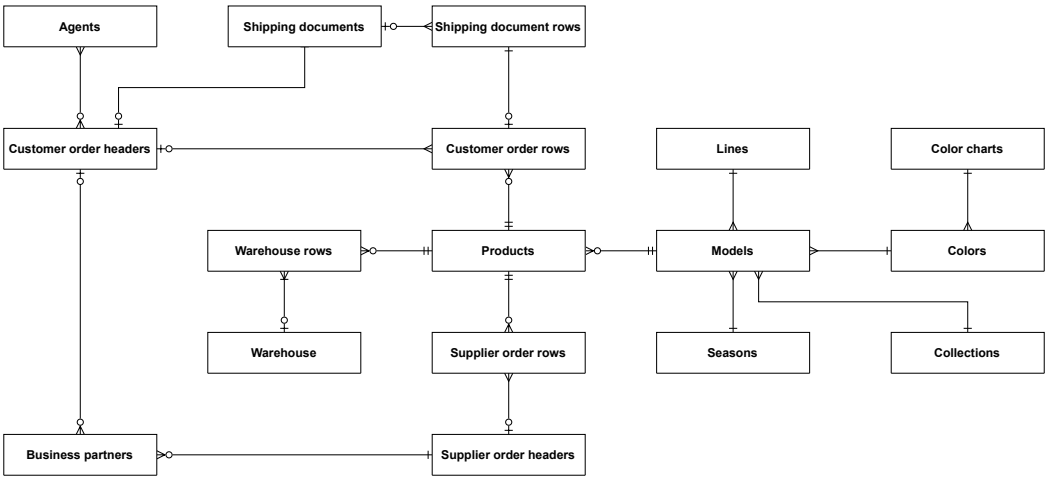


Figure 6.1: Database schema for user tests

[1]https://www.sys-datgroup.com/

6.1. Technical evaluation

Furthermore, several metrics analyses were carried out on the platform's operational performance, including latencies, response time, CPU time, and resource occupation, to assess its temporal optimization. Even though the timing of individual calls may seem of secondary importance, the study confirms that it is crucial for optimizing the system's overall performance.

Moreover, it has been decided to test even further the system capabilities by evaluating the effectiveness of the NLU process - more specifically the schema pruning phase - and the temporal complexity of the most relevant pipeline steps.

6.1.1. Algorithmic complexity

The proposed approach for pruning the schema and generating queries was evaluated in terms of time complexity.

This step is necessary to evaluate the goodness of the prototype and the proposed algorithms. Given the high reusability of the components, it was decided to perform this temporal analysis only for the *schema pruning* and *dashboard creation* process.

Schema pruning This algorithm performs keyword extraction with a temporal complexity of $O(N)$, where N is the number of tables in the schema. After encoding the keywords, it iterates over all tables in the schema and then for each attribute of the table, it calculates the cosine distance between the attribute and the encoded keywords.

The resulting similarity score is stored for each table and attribute. Next, it creates a queue of all tables in the schema and then, while the queue is not empty, it dequeues a table and calculates its weighted sum similarity score based on its neighbors. The algorithm then returns the table with the highest similarity score greater than a certain threshold.

The total time complexity of the algorithm can be expressed as $O(N^2 + N + N \cdot m)$, or just $O(N^2)$, due to the nested for-loops iterating over all tables and the average number of attributes per table (m).

Dashboard creation The algorithm underlying the proposal of the different indicators given a request for a dashboard is based on the selection of relevant and eligible attributes for extraction. The algorithm includes the schema pruning process which reduces the span of interest only on the entities really necessary for the extraction: for this reason, the complexity baseline is greater than the time complexity of the *schema pruning*.

Subsequently, for each eligible table, it is recombined into a query given a fixed number of possible template queries. Finally, for each extracted query, we proceed to the projection (π_M) by iterating on all the relevant columns of the tables involved.

Therefore, being n the number of eligible tables, with $n \leq N$, and m the average number of columns per table, the maximum complexity of the algorithm is $O(N \cdot (1 + m) + N^2 + n \cdot m)$, or just $O(N^2)$.

Both the algorithms suffer from quadratic dependence with the total number of tables in the declared schema N, due to the necessity of selecting the most interesting tables for each user request in order to narrow the span of computation and increase the accuracy.

6.1.2. Response times

To evaluate the efficiency of the entire application, measurements were made on the latencies of each process, in order to evaluate the expected baseline on the temporal performance of the prototype. We evaluated the performance of the developed prototype by computing metrics for the response time, CPU time and the throughput of the main processes. The tests were performed on a machine with 16GB of RAM, utilizing a 12th Generation Intel Core i7-1255U processor running at a frequency of 1.70GHz.

For each component tested and each metric, 30 requests were made for which the following data were collected.

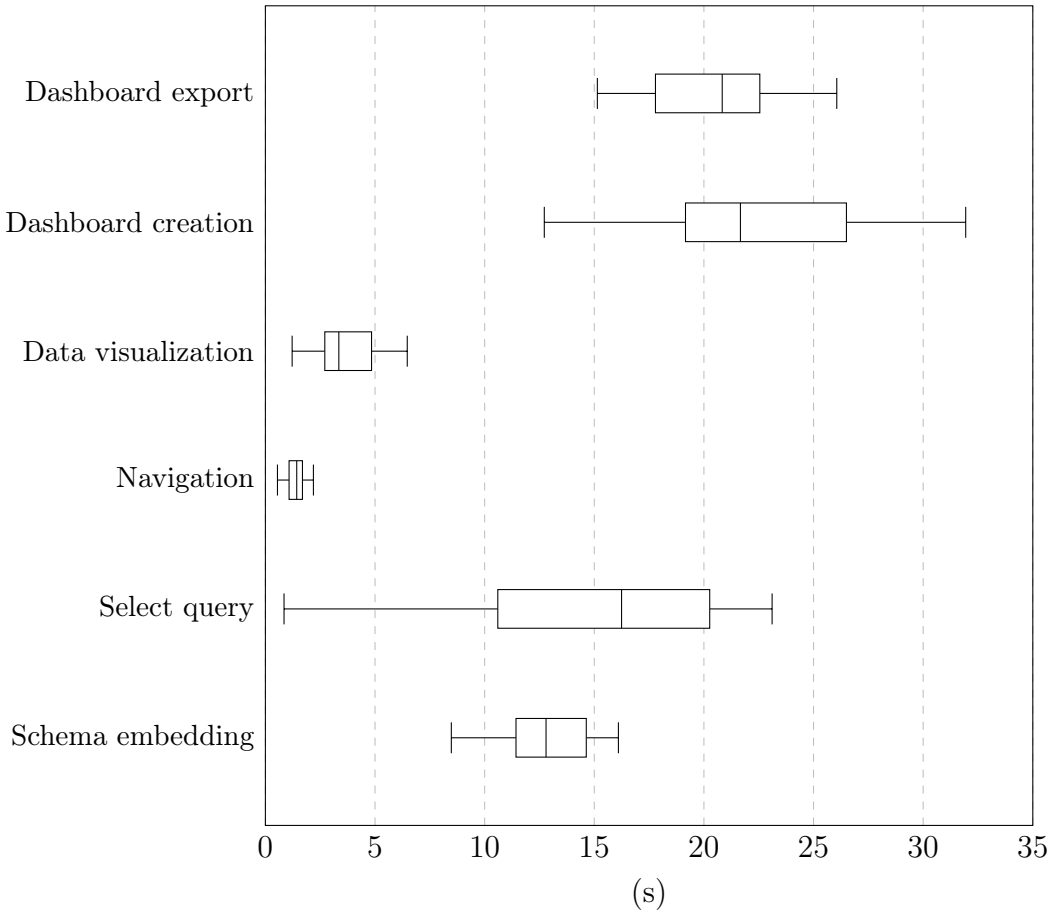


Figure 6.2: Response times

Process	$\overline{r}$(s)	σ_r(s)	X_r	CPU_r(s)
Schema embedding	12.77	2.13	0.07	11.26
Select query	14.56	6.58	0.06	9.78
Navigation	1.39	0.48	0.72	1.21
Data visualization	3.82	1.57	0.26	3.54
Dashboard creation	22.85	5.43	0.04	20.72
Dashboard export	20.69	3.39	0.05	20.28

Table 6.1: Analytical metrics

As we can see, the fastest process is navigation between entities and extractions, with an average response time of 1.39s, while the slowest process is dashboard creation, with an average response time of 22.85s due to parallel data extractions. The standard deviation and throughput vary between processes, with the select query process having the highest standard deviation of 6.58s and navigation having the highest throughput of 0.72 requests served per second.

The CPU time for each process is also reported, with dashboard creation and dashboard export having the highest CPU time of 20.72s and 20.28s, respectively. These results are due to the complexity of the processes, which requires a significant amount of resources to simultaneously generate multiple interconnected indicators and combine the extracted information into a slideshow. In order to decrease the response time, it is possible to incorporate a parallelized approach to eliminate the sequential creation and execution of queries. This strategy has the potential to streamline the process and significantly reduce the overall response time. However, the main difference between the CPU time and the actual response time is mainly due to the REST API inter-exchange and the SQL query execution on the selected data source.

6.1.3. Effectiveness

To validate the proposed infrastructure, a technical analysis was carried out on the NLU precision and accuracy. It should be highlighted that the effectiveness of this process is highly dependent on the accuracy of the text-to-SQL translation powered by GPT and on the schema pruning algorithm (Section 4.5.1). Schema pruning is a mandatory task for any process of transforming natural prompts into SQL code that works on large databases, reducing the span of entities on which to operate. For this reason, it is necessary to evaluate the technical capacity of the schema pruning module through a validation process of the results.

For this purpose, using the implemented prototype, we analyzed 290 user prompts and the related selected tables, comparing them with the expected entities needed to obtain an executable query. The average accuracy of the system settles at 89.31%, with a recall of 90.20%, leading to an F1 score of 0.8975. However, the module's average accuracy in prioritizing which tables to use is 78.37%.

These results demonstrate how the module can highlight most of the relevant results to the activity. Furthermore, the high recall confirms that the algorithm is capable of presenting more relevant results than superfluous entities. This factor is crucial to avoid that, during the query generation, the high number of entities generate noise in the translation, thus it improves the global performance of the system.

However, the algorithm is less efficient in prioritizing the tables - therefore giving ordinal importance to the entities considered relevant - but it is to be considered a good result which is also mitigated by the inferential capacity of the Query generation module (Section 4.5.3).

6.2. User study

Our research aims to define a new approach to enable easier access to information. While a purely SQL-based approach and manual data visualization have greater theoretical potential, the required timescales, skills, and knowledge make this approach more complex.

To evaluate the efficiency and effectiveness of our proposed infrastructure from the perspective of the final users, we designed a study that compared the performance of a sample of testers while interacting with our prototype (Queric) and then with Excel.

We identified a set of tasks that could be executed using both systems and authorized participants to access and execute queries, choose display modes, and group them into dashboards. The selected tasks would allow us to assess the validity of the design choices underlying the main functions of our prototype, in relation to navigating the data source, extracting information conversationally, displaying data visually via verbal commands, and composing dashboards using natural text.

We then evaluated the time taken by the study participants to execute the tasks and their satisfaction.

The evaluation study design was initially tested and refined by involving several pilots that helped us highlight some issues and insights exploited in the final executed user test.

6.2.1. Pilot test

Conducting a pilot test for an evaluation procedure is crucial as it allows for the identification and resolution of potential issues, ensuring a more refined user testing process. Therefore, we conducted a pilot test with 6 participants who are completely unexpert in this domain, composed of 3 females and 3 males.

The participants were introduced to the user interface and the final goal of the application with a 3-minute presentation before starting the test. The test then consisted of 10 tasks that required the participants to use natural language queries to access and analyze data from a structured source. The tasks varied in complexity and difficulty, ranging from simple queries to complex aggregations and comparisons. We measured the success rate of each task, as well as the time taken and the general satisfaction level of the participants.

The participants in the sample had an average age of 26.83 years, with a $\sigma = 12.02$. The age range varied from a minimum of 18 years to a maximum of 51 years. The test results indicated that participants successfully completed an average of 8.5/10 tasks, partially succeeded in 0.3/10 tasks, and experienced 1.17/10 failed tasks.

Considering the sample size and the novelty of the application, these results were encouraging and suggested that the prototype was usable and effective for simple-medium tasks of conversational data access and analysis to structured sources.

Takeaways During the evaluation process, several noteworthy issues have emerged, notably regarding the time invested in carrying out the entire set of tasks. It was observed that the overall average execution time for all ten tasks amounted to approximately 1 hour and 2 minutes. Recognizing the need for optimization, we have taken the initiative to revise and refine the task requirements. As a result, we have decided to reduce the number of requested tasks, narrowing our focus to the 4 activities deemed most relevant and essential.

Furthermore, in an effort to eliminate redundancy and improve task efficiency, we have merged certain previously distinct and repetitive requests into a single consolidated task. Indeed, some of the proposed activities were too simple, thus highlighting the need for a more complex task definition.

By implementing these adjustments, we aim to enhance both the usability and effectiveness of the prototype based on the user test qualitative feedback, while also taking into consideration the quantitative data related to execution times and success rates.

6.2.2. Tests

Participants

We enrolled 12 participants in our study, consisting of 9 males and 3 females who were employed as middle managers or IT project managers in medium to large companies. The sample overview is reported in Table 6.2. The mean age of the participants was 48.42 years (σ = 12.52, min = 28, max = 68). More precisely, the sample is a subset of the user group introduced in Chapter 3.

A demographic questionnaire submitted at the beginning of the study revealed that the participants had an excellent level of experience in IT ($\overline{x}$ = 7.86, σ = 1.30, min = 5, max = 9), chatbot usage ($\overline{x}$ = 6.80, σ = 1.37, min = 5, max = 9), and using SQL language ($\overline{x}$ = 7.40, σ = 1.12, min = 6, max = 9). In terms of data analysis and visualization, the participants exhibited a significant range of abilities, with a solid level of knowledge for analytical tasks ($\overline{x}$ = 7.47, σ = 1.31, min = 5, max = 9) and a slightly weaker level for visualizing information activities ($\overline{x}$ = 6.93, σ = 1.58, min = 3, max = 9).

The results demonstrate a highly heterogeneous dataset, with the Likert scale used to assess skill levels, ranging from 1 to 10 (1 = very low, 10 = very high). Overall, the sample is highly educated, with 5 individuals holding master's degrees, 4 with bachelor's degrees, 2 with a Ph.D., and 3 high school technicians.

Participation code	Age	Gender	Role
P1	68	M	Sales manager
P2	48	M	Development team leader
P3	33	M	Project manager
P4	38	M	BU manager
P5	56	M	BU manager
P6	28	F	Logistics manager
P7	54	M	IT manager
P8	59	M	Sales manager
P9	62	M	Project manager
P10	38	F	Development team leader
P11	42	M	Product owner
P12	55	F	BU manager

Table 6.2: Test participants

Tasks

The participants are required to use the system to complete 4 tasks reported below (Table 6.3) to better and fully cover the functionalities exposed by the prototype. The tasks must be performed in the Chat component.

Request	Type
Extract some sales orders and visualize it	2
Try to extract the total sold per customer	1
Try to find the best-selling product for sale season *estivo*	1
Extract each day of the last year by reporting the daily total sold	2

Table 6.3: Tasks

Each task refers to a specific category of activity.

1. The first category involved two tasks that required the users to find specific data items by applying selection, projection, join, and filtering criteria.
2. The second category involved two tasks that asked the users to extract the maximum inferential power from the application, guiding the prototype to the expected result.

For each task, users had a maximum time of 3 minutes (180 seconds). In accordance with the within-subjects design, each participant performed all the 4 tasks using the prototype, followed by the same series of activities using prepared raw extractions in Excel.

Procedure

The research was conducted remotely via web call and RDP (Remote Desktop) on a preconfigured virtual environment that blocked outgoing internet requests using a firewall. The connection was limited to OpenAI hosts via a whitelisted connection. The entire study was recorded and reported.

No facilitator was present during the study, and the application program was not illustrated. However, each user was introduced to the database on which they operated. Despite not having specific expertise in the scheme under analysis, all users had good knowledge of the domain, which was a seasonal commercial cycle.

In total, the study lasted for three days. During the study, each user connected to the RDP, and the supervisor monitored their work to prevent external interference.

The participants had previously completed a demographic questionnaire integrated with a survey on technical-specific skills in the IT and analytical fields. Then, each participant was invited to carry out the experimental tasks independently for each system tested - firstly the Queric prototype and then Excel. At the end of all the experimental activities with each system, the participant completed an online questionnaire regarding the systems used.

Data collection

Data collection is a relevant aspect of any research project, especially when it comes to evaluating usability. In this context, usability pertains to the degree to which a product or service can be utilized effectively, efficiently, and satisfactorily by its designated users. [30]. To measure usability, researchers need to collect data from users through various methods, such as surveys, interviews, observations, experiments and tests.

The choice of data collection method depends on the research questions, the type of data needed, and the resources available.

One of the challenges of data collection for usability is ensuring data quality and data usability. Data quality refers to the accuracy, consistency, timeliness and conformity of the data, while data usability refers to the degree to which the data can contribute to actionable insights and reduce the risk of misinterpretation or misuse.

It is therefore crucial to collect high-quality and usable data that can help to evaluate usability in terms of *effectiveness*, *efficiency* and *satisfaction*, following the software engineering standard ISO/IEC 9126 [31].

Effectiveness refers to the accuracy and completeness of users' tasks, efficiency refers to the resources expended by users to achieve their goals, and satisfaction refers to users' subjective feelings about their experience [30]. These three dimensions of usability can provide valuable feedback for improving products or services and enhancing user satisfaction.

One of the methods to evaluate the usability of a system or a product is to conduct usability tests with potential users, measuring both quantitative and qualitative aspects of user experience, such as task completion time, error rate, user satisfaction, and user feedback. Two common tools for measuring user satisfaction and perceived usability are the System Usability Scale (SUS) and the NASA Task Load Index (NASA-TLX).

More precisely, the SUS [32] is a post-test questionnaire that consists of 10 items rated on a 5-point Likert scale. The SUS provides a global score of system usability, ranging from 0 to 100, with higher scores indicating better usability. The SUS is widely used and validated across different domains and platforms.

The NASA-TLX is a post-task questionnaire that assesses the mental, physical, and temporal demands of a task, as well as the effort, performance, and frustration levels of the user. The NASA-TLX provides a multidimensional measure of perceived workload, which can reflect the complexity and difficulty of a task. The NASA-TLX is also widely used and validated in various contexts, especially for mission-critical tasks [33].

Indeed, the combination of SUS, NASA-TLX, and qualitative data can provide a comprehensive evaluation of the usability of a system or product. SUS and NASA-TLX can capture the overall impression and workload of the user, while qualitative data can reveal the specific issues and opinions of the user. Indeed, by triangulating these data sources, usability testers can identify the system's strengths and weaknesses, and suggest improvements for future iterations.

6.2.3. Analysis

To evaluate the effectiveness of our approach, we examined qualitative data gathered from user tests, which includes feedback obtained through methods such as LSA (Latent Semantic Analysis), and quantitative data regarding execution times and success rates to provide a comprehensive evaluation of the prototype's usability and performance.

Usability analysis

The overall system has been tested and compared to one of the most efficient and well-known applications for reporting - i.e. Excel. Thus, usability has been evicted by comparing the time spent by users accomplishing the assigned tasks on both systems and the satisfaction perceived. More specifically, following what was proposed in previous works [2] [3], we divided the analysis into *efficiency*, *effectiveness* and *satisfaction*.

Efficiency The system's efficiency, defined as the capability of performing a task involving a lower quantity of resources, has been evaluated by considering the time each user spent performing the activities.

The overall results are encouraging (Queric $\overline{x} = 124.58s, \sigma = 14.71$, Excel $\overline{x} = 146.06s, \sigma = 19.67$) highlighting how, in the same amount of time, users are able to generate correct results and refine them more naturally.

The results are not statistically different as demonstrated by a compared paired t-test ($t = -2.281, p = 0.063$).

	$\overline{x}_q$(s)	σ_q(s)	$\overline{x}_e$(s)	σ_e(s)	T	df	p
Task 1	130.50	55.44	127.58	39.03	0.165	12	0.564
Task 2	103.92	64.52	134.17	47.41	-1.298	12	0.110
Task 3	125.67	45.48	150.83	35.30	-1.572	12	0.072
Task 4	138.25	55.57	171.67	17.60	-1.824	12	0.048

Table 6.4: Paired Sample single negative tail T-Test results on execution times

Table 6.4 shows the user time performances divided by task and the results of the paired sample single-tail t-test. As demonstrated by the p-value, Tasks 3 and 4 - the most difficult ones - are statistically different with good significance. This condition is easily explainable due to the fact that constraining at 3 minutes the maximum amount of time per task, the usage of a completely-graphic tool - i.e. Excel - may require greater timespans. However, it is clear as, once one understood the task and the operation to perform, the usage of Excel could asymptotically achieve higher performances. For the other two tasks, this is not completely true, and, even if the performances award our system, the difference is not statistically significant.

By analyzing the distributions of timing across the different participants, it is clear as the sample considered is technically varied, manifesting different performances. More specifically, we can appreciate a skewed bimodal distribution for tasks 1, 3 and 4, demonstrating an internal difference among testers' skills. To this extent, Figure 6.3 allows us to visually analyze the distribution of data using the frequency of occurrences in 15 bins.

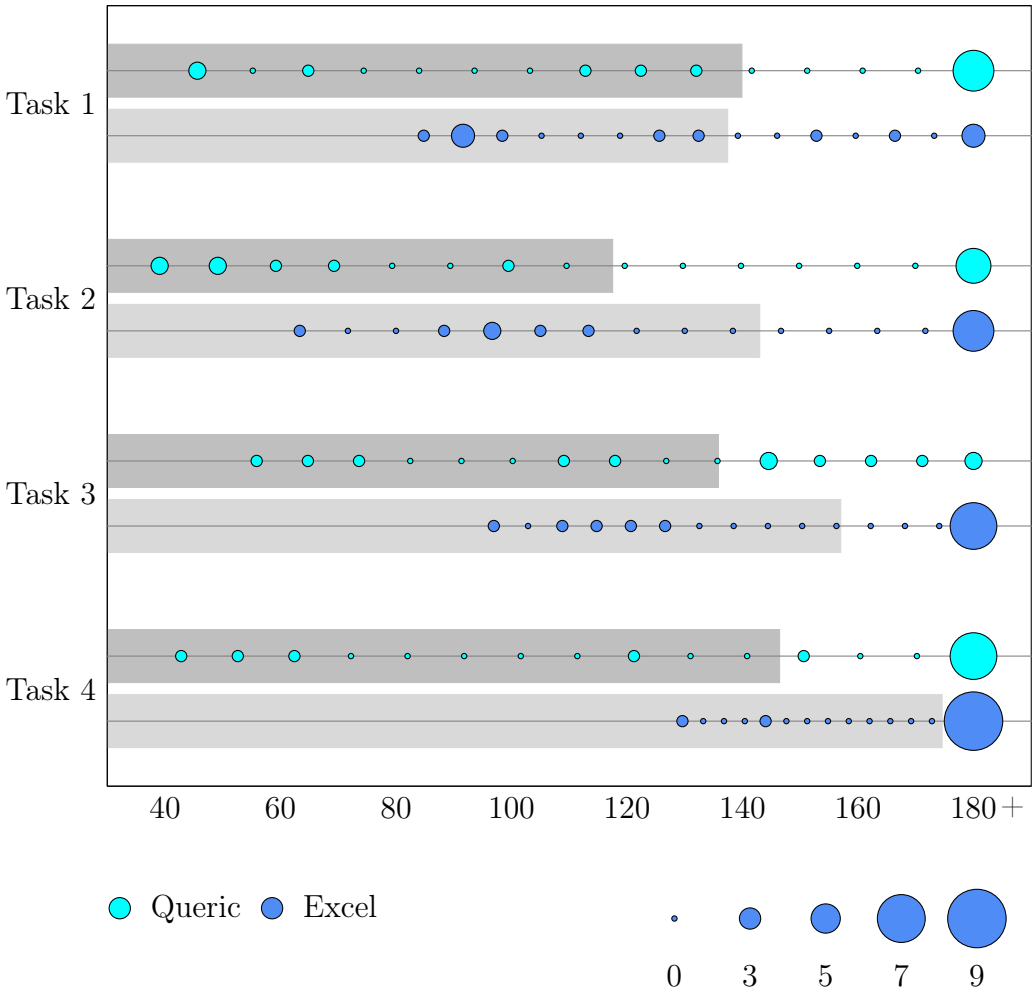


Figure 6.3: Execution times distribution/frequencies and average values defined over 15 intervals grouped by system and task

Effectiveness To assess the effectiveness of the system, we utilized a success rate calculation which is commonly used to gauge how well systems aid in accomplishing tasks. Our study involved categorizing each query performed by participants as "Success" if the task was fully completed, "Partial success" if some results were obtained but without respecting completely the assigned task, and "Failure" if the task execution was incorrect. Thus, the success rate was then computed as the percentage of users who were able to successfully complete the tasks, weighing half the partially successful executions - i.e. $0.5 \cdot \#partial$.

Following this definition, Queric obtained a success rate of 73.95% over 48 tasks (Success = 30, Partial success = 11, Failure = 7), against a success

rate of 64.58% (Success = 23, Partial success = 16, Failure = 9) for the tasks executed with Excel. The detailed overview is reported below in Table 6.5.

By analyzing the results is clear that some tasks are balanced and the difference is not statistically significant as demonstrated by the Wilcoxon signed-rank test (Z = -0.051, p = 0.4806).

Qualitatively, the longer the task is, the more effective the conversational access appears in the same time span, allowing for a higher number of attempts.

	Queric			Excel		
	Success	Partial	Failure	Success	Partial	Failure
Task 1	6	3	3	7	4	1
Task 2	8	2	2	7	2	3
Task 3	10	1	1	5	5	2
Task 4	6	5	1	4	5	3

Table 6.5: Results grouped by success

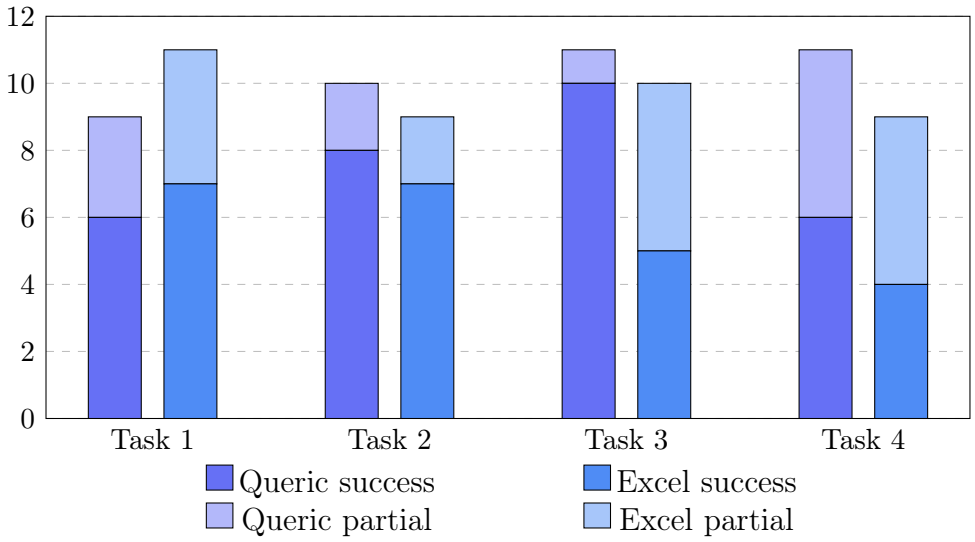


Figure 6.4: Successful and partially successful tasks grouped by system

Satisfaction The analysis of satisfaction is crucial to understand how effective both systems are perceived by users. To this extent, the test involved SUS and NASA-TLX questionnaires, leading to the following results.

Satisfaction through Usability In order to support the usability analysis, a SUS questionnaire has been submitted by each user at the end of the practical execution for each system.

Focusing on the conversational system, it has demonstrated good usability measured by means of an average SUS score of 70.19 ($\sigma = 8.53$), considering the average assumed score of 69.5 as highlighted in [34]. However, even Excel has received extremely positive satisfaction reviews, leading to an average SUS score of 74.89 ($\sigma = 7.84$). We need to consider that Excel is a consolidated, known and successful complete application whose fame clearly justifies its performance.

As suggested in [3], the SUS questionnaire has been divided into three parts: System Learnability (questions #4, #7, #14,#15), Results Quality (#21, #22, #23, #25, #26) and System Usability (the remaining questions). The Learnability has been quantified insufficient for Queric ($\overline{x} = 65.83$, $\sigma = 16.89$) and in line with the standards for Excel ($\overline{x} = 67.22$, $\sigma = 13.02$). Considering the Quality of the results, both systems are good, with Queric more capable of impressing users (Queric $\overline{x} = 78.70$, $\sigma = 10.96$; Excel $\overline{x} = 70.37$, $\sigma = 10.71$). Finally, the overall System Usability has been estimated as positive for both systems, but with an outperforming result for Excel (Queric $\overline{x} = 72.89$, $\sigma = 8.15$; Excel $\overline{x} = 78.40$, $\sigma = 9.09$).

Satisfaction through Workload To test the workload experienced during the test, the final questionnaire embedded the NASA-TLX points. The results prove that the conversational approach is slightly more demanding ($\overline{x} = 54.89$, $\sigma = 10.06$) than Excel ($\overline{x} = 48.23$, $\sigma = 10.13$). However, the t-test unveils that the results are not statistically significant ($t = -0.98$, $p = 0.342$).

	$\overline{x}_q$	σ_q	$\overline{x}_e$	σ_e	T	df	p
Mental Demand	49.83	18.76	51.00	16.22	-0.261	12	0.796
Temporal Demand	87.16	10.20	51.87	20.76	3.924	12	0.013
Performance	56.83	23.73	68.5	16.60	-1.315	12	0.205
Effort	51.00	16.60	38.75	20.48	0.564	12	0.581
Frustration	52.16	27.83	44.875	26.11	0.892	12	0.382
Physical Demand	32.33	24.68	34.375	20.61	-0.755	12	0.461

Table 6.6: Paired Sample single negative tail T-Test results on workloads

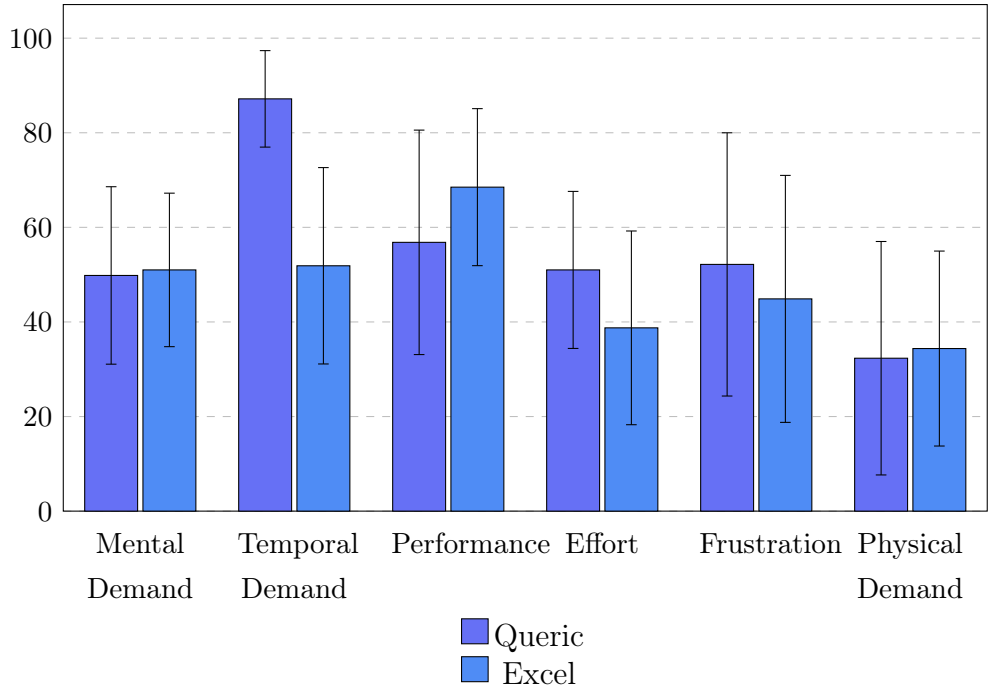


Figure 6.5: Workload of both systems by category

Qualitative analysis

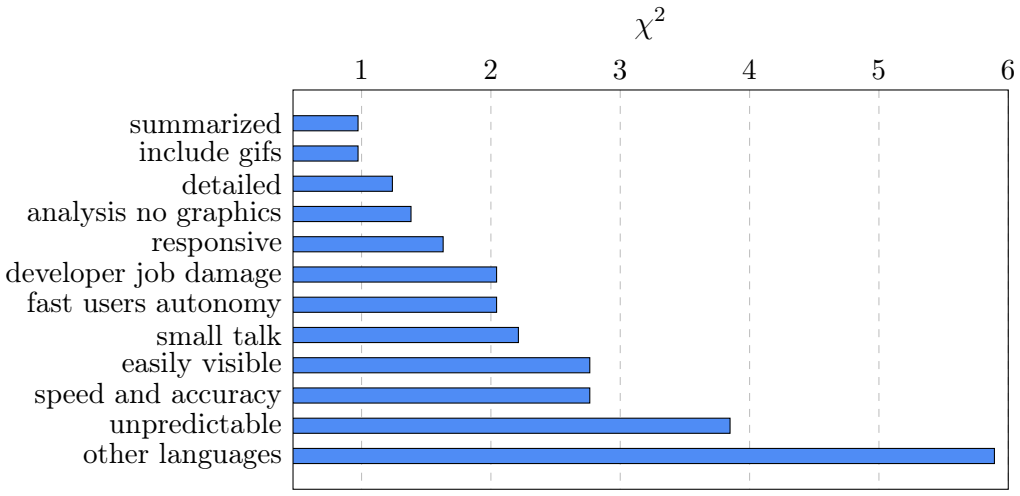


Figure 6.6: Top LSA Chi-squared Feature Selection

The results of the qualitative test comparing a conversational agent for accessing data to Excel are quite interesting. Users reported that Queric is easy, flexible, and fast to use, which allows for a conversational approach to accessing business data. User P4 shared that *"it is easy and simple to interact with, but also extremely satisfying in achieving what's requested"*. This feedback indicates how the conversational interface has improved user experience and engagement with the system.

As highlighted, the responses can appear unpredictable and unforeseeable, resulting in a sensation of uncontrollability. This feedback suggests that there may be room for improvement in the system's natural language processing capabilities.

User P1 suggested adapting the system to virtual assistants, stating that *"I see great use of the app integrated with virtual assistants"*. This feedback highlights the need to consider user preferences and device compatibility when designing conversational interfaces.

Users also suggested several improvements to the system to enhance its functionalities, including the addition of more graphics and charts, allowing for more conversational interactions, and understanding other languages than English. One user recommended adding the radar chart, while another user suggested making the access to business data more conversational.

Additionally, user P12 proposed that the system should be able to understand less detailed requests, as they mentioned, *"to make yourself understood by the bot, the formulation of requests must be detailed and a little too accurate"*.

The qualitative results demonstrate that our proposal has advantages over traditional tools such as Excel, but there are still areas for improvement. Despite the limitations reported by users, the system's flexibility, ease of use, and speed make it an attractive option for scenarios such as sales analysis, reporting, and drill-down of information. By taking user feedback into account and continuously improving the system's functionalities, conversational agents have the potential to revolutionize the way companies access and interact with data.

LSA analysis To extract topics from the open answers provided by the users, we used Latent Semantic Analysis (LSA), a technique that identifies the relationships between words in a set of documents to highlight underlying topics. First, answers were preprocessed by removing stop words and stemming the remaining words. Then, a term-document matrix is created, where each row represents a word, and each column represents a document. We then applied singular value decomposition (SVD) to reduce the matrix dimensionality and identify the most important topics.

We can appreciate that *unpredictability* is a topic that is relevant to the open answers. However, the analysis also demonstrates a great appreciation for results and usability, which are also important topics in the feedback provided by the users. This information is highlighted in Figure 6.6, where the most relevant topics are shown in a visual representation. The results of our test of the conversational agent for accessing data showed that many users were impressed by the system's automated chart creation, summarization, and dashboard features. These functionalities seemed to spark a sense of wonder among users who were delighted to see their data easily transformed into visualizations and summaries.

Notwithstanding, we also noticed that some users experienced difficulties in clearly communicating their requests to the system, highlighting the need for continued development to improve the system's ability to understand and respond to user requests accurately. The positive reactions are encouraging and suggest that it has the potential to be a valuable tool for users seeking to access and analyze data conversationally and intuitively.

Speed and easy interface One feedback supporting the topic of "Speed" is, *"Sometimes it gets stuck in some processes computing some requests, but it can be resolved by cleaning the chat"* (P8). This feedback highlights the need for concise and summarized reports to improve the efficiency and speed of the application, addressing potential processing issues that could cause frustration.

Regarding "Easily Visible", a user mentioned, *"Make the areas where you can write more easily visible to the user"* (P9). This feedback emphasizes the significance of clear and prominently displayed user interface elements facilitating easy interaction and input. By enhancing visibility, users can readily locate and utilize the necessary features, leading to a more intuitive and user-friendly experience.

Job damage against autonomy Looking at the extracted LSA scores, two contrasting topics that emerged are "Fast User Autonomy" and "Developer Job Damage", highlighting the need for empowering users with fast and autonomous capabilities while considering the potential risks and challenges that developers may face.

One example supporting the importance of "Fast User Autonomy" is the feedback stating, *"Conversational BI and fast data search queries"* (P1). This feedback emphasizes the significance of enabling users to independently access and interact with business data using conversational interfaces, leading to quick and effective decision-making. Users highly value the ability to rapidly retrieve information and gain insights without relying heavily on developer assistance.

In contrast, the topic of "Developer Job Damage" is supported by feedbacks expressing concerns such as *"This could damage developers' jobs"* (P12). These remarks acknowledge the potential challenges that AI and similar advancements pose to developers, as they may need to invest additional effort in developing specialized skills to compete directly against the rise of automation. This feedback also reflects users' anxiety and fear regarding the potential impact of these competence-destroying innovations.

These contrasting topics reflect the importance of striking a balance between empowering users with fast and autonomous capabilities while considering the potential impact on developers.

Therefore, we believe that ensuring user autonomy should be accompanied by clear communication and effective request formulation to mitigate any negative consequences that may arise for the developer's job.

Small talk The feedback, *"Making it even more conversational also in doing small tasks, reaching the complete 'conversationality'"* (P8), reflects the desire for the chatbot or application to engage in casual conversation and handle small tasks.

Socially speaking, small talk is an essential aspect of human communication that serves to establish connections. When it comes to integrating small talk into Queric, we see room to significantly enhance the overall user experience. We believe that, by incorporating small talk capabilities, Queric can engage users in casual and friendly conversations, allowing them to feel more at ease while interacting with the tool. During and after the tests, several users have expressed that having more verbose and relaxed responses, even including some chitchat, would enhance their experience.

Integrating small talk into Queric goes beyond simply being a pleasant addition. It fosters a sense of naturalness and familiarity in the user interface, creating an environment that feels more like a conversation with a friendly companion rather than a robotic information retrieval system. This seamless blend of small talk and information retrieval can make the interaction with Queric more enjoyable and human-like.

Furthermore, users have indicated that breaking the flow of the conversation with occasional small talk adds an element of surprise and variety, making the interaction with Queric more dynamic and enjoyable.

Transparency and unpredictability In the realm of such automated tools, there is a growing concern about transparency regarding the data and retrieval operations that underlie chatbot systems like Queric. As reported in previous sections, numerous users have expressed apprehension regarding the unpredictability of the results generated by these tools.

This caution can be seen as a clear indication that users desire a more comprehensive understanding of the "back office" tasks performed by our prototype.

While users appreciate Queric's ability to infer information, they also value knowing precisely how the system functions and what specific steps it takes to

produce the results presented to them. By providing users with a transparent and detailed overview of the data and retrieval operations and reducing the tool's opacity, Queric can establish trust and address users' concerns, allowing them to make more informed decisions based on their understanding of the tool's processes. Transparency and trust in reporting tools not only meet user needs but also promote accountability and responsible use of AI-driven conversational tools.

6.3. Lessons learned

The experimental findings highlight intriguing insights regarding the performance of Queric and Excel for different tasks. While the results indicate that Queric exhibits greater efficiency for Task 4 and Excel for Task 1, it is relevant to note that the observed differences between the two tools do not reach statistical significance. However, the observed trends provide valuable indications of relative strengths and weaknesses.

Notably, as the complexity of the tasks increases, Queric's efficiency becomes increasingly apparent. This observation can be attributed to the inherent nature of Queric as a fully conversational tool, which does not require users to navigate and interact with a graphical interface. Consequently, in Excel, users may encounter longer completion times due to the additional cognitive load involved in navigating through the large set of features.

On the other hand, Excel's familiarity and reliance on spreadsheet-based calculations make it a more suitable choice for Task 1, where numerical analysis and manipulation are key components. Users accustomed to Excel's interface and functions may experience smoother navigation and quicker execution in such rapid tasks; however, as the complexity of tasks increases, demanding additional manipulation steps, it is noteworthy that Excel takes more time to complete, while Queric does not encounter the same issue.

Moreover, the sequential execution of tasks, following a numerical order, offers valuable insights into the learning effect experienced by users. The use of Queric, a tool unfamiliar to participants before the experiment, leads to a more pronounced learning effect in later tasks.

As users become increasingly acquainted with the features and functionalities of Queric, their efficiency improves, reflecting the positive impact of familiarity and experience on task performance. In contrast, Excel, a tool that participants were already familiar with, does not generate a substantial learning effect throughout the test.

It is worth noting that users have expressed great amazement at the automated extraction of charts and summaries when making data requests. Their excitement has been evident in their remarks about the creation of dashboards and the automated generation of slideshows. Therefore, we consider these elements to be vital and fortuitous factors that should be taken into account when developing conversational access to information. These elements have not only captivated their attention but also provided immense value and convenience in their data exploration and analysis tasks.

Considering the overwhelmingly positive reception and the evident impact these features have had on user experience, it becomes imperative to recognize them as vital and fortuitous factors in the realm of developing conversational access to information.

Zooming out, a deeper analysis of the sample reveals its inherent heterogeneity (Chart 6.3), which suggests the presence of at least two distinct user subgroups with varying abilities to execute the assigned tasks. The effectiveness and efficiency of each tool can be influenced by factors such as prior experience, cognitive abilities, and learning styles, highlighting the need for tailored approaches to maximize user performance.

By understanding the diverse range of users and their specific requirements, developers can fine-tune the functionality and design of conversational access systems to ensure optimal outcomes for each individual. This user-centric approach, which takes into account the interplay between technical capabilities and human characteristics, could pave the way for a truly transformative and empowering information access platform.

Additionally, the lower efficiency observed with Queric may be attributed to the challenges in expressing their intentions clearly and concisely in natural language. While Queric's interface allows for more intuitive interaction, the ability to effectively communicate instructions and queries in a manner easily decipherable by an artificial intelligence system proves to be a crucial factor.

It highlights the significant role of prompt engineering in leveraging the potential of language models - e.g. LLMs. Crafting well-designed prompts that elicit precise and unambiguous responses becomes increasingly essential to enhance the efficiency and accuracy of interactions with language models like Queric.

In conclusion, the experimental findings reveal the nuanced dynamics between Queric and Excel for different tasks. While Queric demonstrates higher efficiency for more complex tasks, Excel remains a favorable choice for numerical analysis. Understanding the impact of users' familiarity, the sequential progression of tasks, and the role of prompt engineering contributes to optimizing the utilization of these tools and ultimately enhancing overall task performance.

The conducted test also enabled a direct comparison between the prototype and one of its toughest competitors. According to the results, there are no significant disparities in terms of usability, satisfaction, and effectiveness. Numerous comparisons suggest that the prototype and its rival are on par across multiple aspects, with Queric excelling in efficiency and serendipity while Excel shines in matters of completeness and integration.

However, the use of conversational AI in business information can arise some issues of security. Indeed, data security is of paramount importance when it comes to extracting business operational data, necessitating strict measures to ensure the protection and privacy of sensitive information. Limiting access privileges and avoiding sharing data with external services helps maintain control and confidentiality, safeguarding the integrity of the organization's valuable data assets.

These findings indicate that the prototype performs well, especially considering that experienced users of Excel, who heavily rely on information, consider Queric a valid and captivating alternative. It can be attributed to the conversational approach, which, in a broad sense, ensures:

- The ability to infer information
- High-quality results and a lower level of technical complexity
- Natural, precognitive and human interactions
- A more democratic access and distribution to knowledge

These considerations, supported by semantic analysis (LSA), provide great optimism regarding the technical and commercial prospects of employing a conversational approach to information. The ability to engage in natural language conversations with intelligent systems, coupled with the capacity to extract relevant insights and provide contextually appropriate responses, can significantly enhance decision-making and problem-solving across a wide range of domains.

The promising potential of this approach could unlock unprecedented levels of convenience, productivity, and innovation in the realm of information access, fueling optimism for its adoption, not only by enhancing the overall user experience but also by opening up encouraging avenues for various commercial applications, such as customer support, data analysis, and decision-making assistance.

7 | Conclusions and future developments

Conversational intelligence has gained significant traction in various domains representing a crucial step forward in human-computer interaction and, when applied to data analysis, allows users to extract insights from data by simply conversing with the system, simplifying the analysis process and enhancing the accuracy and speed of data analysis.

In this Thesis, we proposed an integrated and novel architecture and a paradigm capable of exploiting conversational medium to access and elaborate structured data, extract insights, and generate comprehensive data-driven interfaces.

Initially, we identified the state-of-the-art in conversational intelligence evolution and its application in data analysis. We have highlighted the lack of a comprehensive and agnostic environment that allows natural language access to structured data. To address this gap, we collected requirements from business managers and end-users, with a focus on serendipity, simplicity of design, automation, and visualization of results. Based on these requirements, we developed an infrastructure of different pipelines to support individual data access and analysis and complex data presentation.

Our system provides an automated process for extracting the schema, supporting the enlargement of the schema annotation, and pruning large schema. We generate SQL queries and provide data summarization and visualization based on the context. We also emphasized the inferential power of the application, capable of generating automated dashboards and presentations based on a simple topic. The system finally allows unexpert users to configure the UI, through pure-conversational prompts and simple front-end tasks.

To demonstrate the effectiveness of our proposed application, we developed a prototype, described the technologies, and provided a detailed implementation of the product. The prototype can handle conversational commands, including also features such as automated data extraction and presentation.

Aware of the continuous evolution in the field of conversational intelligence, we have deemed it crucial to give space to one of the most efficient models to date in the management of natural language, i.e. the GPT model. Its ability to process unstructured idioms and highly formalized syntax (code) is not negligible and for this reason it was decided to integrate it into the development rather than compete directly with it.

In the end, we carried out a comparative study between our conversational platform and Excel outcomes. The study encompassed both quantitative and qualitative analyses of interactions, yielding positive findings regarding our approach and revealing opportunities for further enhancements. The results demonstrated that our proposed system offers a more user-friendly method for data analysis, leading to reduced analysis time and improved result accuracy.

In conclusion, our proposed application provides a comprehensive and agnostic environment for natural language access to structured data. The system automates data extraction and simplifies the analysis and presentation process, allowing for serendipity, simplicity of design, automation, and visualization of results. Our comparative study demonstrates the effectiveness of our approach and provides insights for future research. This system has significant potential for enhancing data analysis in various domains and can be further improved by integrating new technologies and features.

7.1. Limitations

While defining our framework, we made several structural and design choices that significantly impacted the system limitations. One of the most significant challenges of a conversational approach to information is data privacy and access privileges. Undoubtedly, given the increasing amount of data being collected and stored, ensuring that only authorized personnel have access to sensitive information is of utmost importance if considering a business scenario. Therefore, it is crucial to ensure proper security protocols to safeguard the data and authorized access to information.

Another limitation that can hinder the effectiveness of a conversational access system is the usage of BLOBs (Binary Large Objects). These large data files can slow down the system and make it difficult to access large amounts of data in a feasible time. Moreover, BLOBs require special handling, making it challenging for conversational systems to access and process the data within these files. The presented conversational system is unable to process images as part of a query, nor can it provides image-based results. This limitation can restrict the system's ability to provide a complete and comprehensive view of the data, especially when dealing with visual data such as images - commonly used in modern ERPs.

Another significant factor to be considered is that conversational inquiries require a try-and-error approach that can be time-consuming and frustrating. NQL allows users to ask questions in a natural language format, but it can be challenging for the system to interpret the user's intent accurately.

The prompt-query-result system that we employed is prone to higher latencies as the schema size and data volume increase. Whether in the form of results or an error message, the user may have to wait several seconds before receiving a response when requesting an output.

Moreover, our schema pruning approach, based on entity semantics, could potentially reduce the system's scalability. For large databases, this operation can be particularly expensive and, when combined with the prompt-query-result runtime process, can make it challenging to apply the system to big data sets and schemas with high cardinality.

In our efforts to create a comprehensive system, we attempted to keep the graphical interface minimal. Nevertheless, our tests uncovered potential complications, especially for users who are accustomed to the extensive array of capabilities provided by contemporary corporate information systems. The lack of clarity regarding which operations are to be performed conversationally and which require manual input can lead to significant confusion, representing a relevant system's limitation.

A final challenge is the difficulty of presenting data effectively and clearly. This factor is especially evident when dealing with complex data sets, impacting the overall application performance and serendipity.

7.2. Further development

Conversational agents have come a long way in recent years, thanks to advances in machine learning and natural language processing. With the development of GPT-based applications such as PandasAI [35] - enabling conversational access to DataFrame - conversational agents now can query data frames using natural language. We believe there is still much potential for further development and improvement of these agents. Moreover, the emergence of ChatGPT plugins [36], which are API services designed to disrupt and enhance the capabilities of GPT models, presents a tremendous opportunity to revolutionize the impact of large language models. Through the ability to fine-tune OpenAI's models for specific functions, these plugins have the potential to significantly expand the range of applications and possibilities for LLMs, pushing the boundaries of their capabilities.

Focusing on our proposal, a key area for improvement is the immediate selection of resources. As previously mentioned, the application currently has the capability to handle multiple outgoing connections to various data sources independently. However, a more integrated approach is necessary to ensure that the agent selects the appropriate resources at the appropriate times. This integration involves combining multiple data sources and facilitating the rapid and accurate processing and analysis of extensive datasets. Consequently, the agent will be better equipped to deliver precise and relevant responses to user queries.

Another area for improvement is automated and enhanced schema annotation which involves annotating the schema of data sources with information that can support entity selection and source pruning. As a result, the application can enjoy improved performance in better understanding the user's queries and providing more accurate and relevant responses.

Furthermore, through the integration of external data mining modules, the agent can extract a greater wealth of information from diverse sources, thereby gaining a deeper comprehension of the user's requirements and preferences. This capability offers valuable descriptive and generative insights into various observed phenomena, signifying a groundbreaking shift in how experts can access and extract knowledge from structured sources. Additionally, this advancement empowers the agent to offer more personalized and impactful recommendations and suggestions.

Moreover, deeper integration of open data and associative data sources, such as ROLAP, can also lead to significant improvements in the agent's capabilities but it will require fusing more advanced techniques of accessing semi-structured resources based on different paradigms. In previous chapters, we have demonstrated how the application can access associative structures such as JSON and, by combining them with new tools such as PandasAI for data frames, can easily extend its capabilities to non-relational data sources as well.

The requirements elicitation has highlighted the relevance of integrating voice and virtual assistant features into a text-based conversational platform, underlining the possibility of making data analysis more user-friendly. Consequently, we have tried to introduce speech-to-text (STT) and, conversely, text-to-speech (TTS) as a simple voice interface to expand the accessibility of our platform, obtaining good results. However, further developments are required, for example focusing on advanced voice dictation techniques [37]. Indeed, our efforts, mainly concentrated on a comprehensive system definition, ran into some renowned STT problems, such as literal dictation of precise parameters and names.

For these reasons, further efforts could concentrate on implementing a more robust and adaptable voice interface. This approach is particularly beneficial for individuals unfamiliar with traditional data analysis tools. By speaking their queries and commands, users can make data analysis more intuitive and accessible. This integration is expected to have a significant impact on the future of data analysis by providing a more efficient and convenient way of working with data.

The continuous development of prompt-based LLMs, exemplified by OpenAI's GPT, promises to significantly enhance the agent's ability to interpret and translate prompts into queries in the coming years. As these models evolve, we anticipate a steady improvement in the accuracy and effectiveness of the results, driven by a deeper understanding of the user's intentions and context. There are reasons to believe that this advancement in LLMs will contribute to more precise and relevant responses, further improving the conversational experience and utility of the agent.

In conclusion, conversational agents have made significant strides in recent years, also combined with automated and AI-mediated data analysis, but there is still ample potential for further development and enhancement. By improving on-the-fly resource selection, enhanced schema annotation and deeper integration of open and associative data sources, conversational agents can become even more powerful and effective tools for querying and interacting with complex data sets.

With the potential to seamlessly access and interact with knowledge, these agents hold the key to transforming how we engage with information. As we venture forward, it is not only the technology that will evolve, but also our own relationship with knowledge, as we embark on a journey of exploration and discovery facilitated by the power of conversation.

Bibliography

[1] Deep Talk. "80% of the world's data is unstructured". In: *Medium* (2021). URL: https://deep-talk.medium.com/80-of-the-worlds-data-is-unstructured-7278e2ba6b73.

[2] Nicola Castaldo et al. "Conversational Data Exploration". In: *Web Engineering - 19th International Conference, ICWE 2019, Daejeon, South Korea, June 11-14, 2019, Proceedings*. Ed. by Maxim Bakaev, Flavius Frasincar, and In-Young Ko. Vol. 11496. Lecture Notes in Computer Science. Springer, 2019, pp. 490–497. DOI: 10.1007/978-3-030-19274-7_34. URL: https://doi.org/10.1007/978-3-030-19274-7%5C_34.

[3] Nicola Castaldo. "A conceptual modeling approach for the rapid development of chatbots for conversational data exploration". MA thesis. Politecnico di Milano - Scuola di Ingegneria Industriale e dell'Informazione, Apr. 2019.

[4] Facebook AI. "State-of-the-art open source chatbot". In: (2021). URL: https://ai.facebook.com/blog/state-of-the-art-open-source-chatbot/.

[5] Senseforth.ai. "How chatbots use NLP, NLU, and NLG to create engaging conversations". In: (2021). URL: https://www.senseforth.ai/conversational-ai/chatbots-create-engaging-conversations/.

[6] AIMultiple. "50+ Chatbot Companies To Deploy Conversational AI in 2023". In: (2023). URL: https://research.aimultiple.com/chatbot-companies/.

[7] OpenAI. "GPT-4". In: (2023). URL: https://openai.com/research/gpt-4.

[8] Microsoft Research Blog. "Conversations with data: Advancing the state of the art in language-driven data exploration". In: (2022). URL: https://www.microsoft.com/en-us/research/blog/conversations-

with-data-advancing-the-state-of-the-art-in-language-driven-data-exploration/.

[9] T.B. Brown et al. "Language Models are Few-Shot Learners". In: *arXiv preprint arXiv:2302.11054* (2023).

[10] Unal Aksu et al. "An Approach for the Automated Generation of Engaging Dashboards". In: (Oct. 2019), pp. 363–384. DOI: 10.1007/978-3-030-33246-4_24.

[11] Luciano Floridi and Massimo Chiriatti. "GPT-3: Its Nature, Scope, Limits, and Consequences". In: *Minds and Machines* 30 (2020), pp. 681–694.

[12] *Google Dialogflow*. URL: https://dialogflow.cloud.google.com/ (visited on 04/2023).

[13] *Amazon Lex*. URL: https://aws.amazon.com/lex/ (visited on 04/2023).

[14] *IBM Watson Assistant*. URL: https://www.ibm.com/cloud/watson-assistant/ (visited on 02/25/2023).

[15] *Facebook's Wit.ai*. URL: https://wit.ai/ (visited on 04/2023).

[16] *Microsoft Azure Bot Service*. URL: https://azure.microsoft.com/en-us/services/bot-service/ (visited on 04/2023).

[17] Tableau. *Tableau*. URL: https://www.tableau.com/ (visited on 04/2023).

[18] Elastic. *Elasticsearch Kibana*. URL: https://www.elastic.co/kibana (visited on 04/2023).

[19] Microsoft. *Power BI*. URL: https://powerbi.microsoft.com/ (visited on 04/2023).

[20] Giuseppe Desolda et al. "Rapid prototyping of chatbots for data exploration". In: *BCNC@SPLASH*. 2021, pp. 5–10.

[21] Marius Gassen et al. "Demonstrating CAT: Synthesizing Data-Aware Conversational Agents for Transactional Databases". In: *Proc. VLDB Endow.* 15.12 (Aug. 2022), pp. 3586–3589. ISSN: 2150-8097. DOI: 10.14778/3554821.3554850. URL: https://doi.org/10.14778/3554821.3554850.

[22] Immanuel Trummer. "CodexDB: Synthesizing Code for Query Processing from Natural Language Instructions using GPT-3 Codex". In: *Proceedings of the VLDB Endowment*. Ithaca, NY: Cornell University, 2022.

[23] Lenis R Wong, David S Mauricio, and Glen D Rodriguez. "A systematic literature review about software requirements elicitation". In: *Journal of Engineering Science and Technology* 12.2 (2017), pp. 296–317.

[24] Zahra Shakeri Hossein Abad et al. "ELICA: An Automated Tool for Dynamic Extraction of Requirements Relevant Information". In: *2018 5th International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*. 2018, pp. 8–14. DOI: 10.1109/AIRE.2018.00007.

[25] URL: https://github.com/features/copilot.

[26] Jack G Zheng. "Data Visualization for Business Intelligence". In: *Global Business Intelligence*. Taylor & Francis, 2018. Chap. 6. DOI: 10.4324/9781315471136-6.

[27] Haode Zhang et al. "Fine-tuning Pre-trained Language Models for Few-shot Intent Detection: Supervised Pre-training and Isotropization". In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Seattle, United States: Association for Computational Linguistics, July 2022, pp. 532–542. DOI: 10.18653/v1/2022.naacl-main.39. URL: https://aclanthology.org/2022.naacl-main.39.

[28] OpenAI. "Best practices for prompt engineering with OpenAI API". In: (2023). URL: https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-openai-api.

[29] "The Socket Programming and Software Design for Communication Based on Client/Server". In: *2009 Pacific-Asia Conference on Circuits, Communications and Systems*. IEEE. 2009.

[30] ISO. *ISO 9241-11:1998 Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability*. Tech. rep. 1998. URL: https://www.iso.org/standard/16883.html.

[31] *ISO/IEC 9126: Software engineering – Product quality*. Tech. rep. Standard, replaced by ISO/IEC 25010:2011.

[32] John Brooke. "SUS: A quick and dirty usability scale". In: *Usability Eval. Ind.* 189 (1995), p. 11.

[33] Sandra G Hart and Lowell E Staveland. "NASA-Task Load Index (NASA-TLX); 20 Years Later". In: *Human Factors* (1988).

[34] Aaron Bangor, Philip Kortum, and James Miller. "Determining what individual SUS scores mean: Adding an adjective rating scale". In: *J. Usability Studies* 4.3 (2009).

[35] Gianluca Venturini. *pandas-ai*. `https://github.com/gventuri/pandas-ai`. 2023.

[36] OpenAI. *ChatGPT plugins*. URL: `https://openai.com/blog/chatgpt-plugins` (visited on 05/2023).

[37] Sadeen Alharbi et al. "Automatic Speech Recognition: Systematic Literature Review". In: *IEEE Access* 9 (2021), pp. 131858–131876. DOI: `10.1109/ACCESS.2021.3112535`.

A | Appendix A

A.1. Elicitation questionnaire

Participant code: _____

1. How frequently do you use dashboard or tabulated data, charts and pivots?
2. How data and information are necessary and crucial for your operations, thus how much data-intensive is your job?
3. How do you feel if you'd be able to change, customize and create data extractions and reports on your own?
4. How frequently do you use operational/transactional data?
5. Have you ever used a conversational interface to interact with data or analytics systems? If so, can you describe your experience?
6. How important is it for you to be able to access data and analytics insights through natural language queries?
7. Do you currently have any data analysts or scientists on your team? If so, how do they typically interact with business users to gather requirements and deliver insights?
8. Can you describe your current process for accessing and analyzing data? Are there any pain points or bottlenecks in this process?
9. How would you feel about a tool that allows business users to ask questions in natural language and receive insights without needing to know SQL or other query languages?
10. Are there any specific features or capabilities you would like to see in a conversational analytics tool?

11. How would you see a conversational analytics tool fitting into your team's existing workflows and processes?
12. How would you measure the success of a conversational analytics tool within your organization?
13. What are your expectations for the speed and accuracy of insights generated by a conversational analytics tool?
14. Are there any concerns you have about implementing a conversational analytics tool, such as data security or governance issues?

A.2. Demographic questionnaire

Participant code: _____

Age: _____

Gender:

Male	Female	Other
☐	☐	☐

Education:

High school	Bachelor	Master	PhD	Other
☐	☐	☐	☐	☐

How do you evaluate your expertise with information technology?

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

How do you rate your expertise with the use of chatbots?

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

How do you rate your expertise with querying databases?

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

How do you rate your expertise with data analysis?

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

How do you rate your expertise with data visualization?

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

A.3. User test questionnaire

A.3.1. Usability

I think that I would like to use this system frequently.

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

I found the system unnecessarily complex.

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

I thought the system was easy to use.

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

I think that I would need the support of a technical person to be able to use this system.

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

I found the various functions in this system were well integrated.

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

I thought there was too much inconsistency in this system.

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

I would imagine that most people would learn to use this system very quickly.

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

I found the system very cumbersome to use.

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

I felt very confident using the system.

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

I needed to learn a lot of things before I could get going with this system.

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

A.3.2. Cognitive effort

How mentally demanding were the tasks?

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

How physically demanding were the tasks?

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

How hurried or rushed was the pace of the tasks?

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

How successful were you in accomplishing what you were asked to do?

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

How hard did you have to work to accomplish your level of performance?

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

How insecure, discouraged, irritated, stressed and annoyed were you?

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

A.3.3. General questions

How easy was it to explore the database with this system, for example to identify the information contained in it or to navigate among the various tables?

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

How easy was it to find the requested information?

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

How easy was it to understand or remember the commands to use in order to find the requested information?

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

How do you consider the way in which the commands are presented?

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

How do you consider the way in which results are visualized?

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

How do you consider the quality of charts?

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

How do you consider the way in which results are summarized?

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

How easy was it to interact with the chatbot?

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

How easy was the extraction of the dashboard?

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

How serendipitous do you consider the overall system?

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

How do you consider the quality of the export?

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

How likely are you to use this tool in your work activities?

Low	1	2	3	4	5	6	7	8	9	10	High
	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	

Which scenarios of use do you think are the most appropriate for the system you used?

Do you have any ideas on how to improve the presentation of results (for example through graphics)?

Do you have any ideas on how to enhance the functionalities of the system?

Main advantages of the system used.

Main disadvantages of the system used.

A.4. Final Questionnaire

With respect to their UTILITY, order the systems you used (1 = the one you consider most useful, 2 = the one you think is least useful):

Queric	1 ☐	2 ☐
Excel	1 ☐	2 ☐

With respect to their COMPLETENESS, order the systems you used:

Queric	1 ☐	2 ☐
Excel	1 ☐	2 ☐

With respect to their EASE OF USE, order the systems you used:

Queric	1 ☐	2 ☐
Excel	1 ☐	2 ☐

Overall, which system would you adopt in your activities?

Queric	1 ☐	2 ☐
Excel	1 ☐	2 ☐

B | Appendix B

B.1. Inquiry process body response scheme

```
{
    "pruning": <associative object of entry table-
       similarity>,
    "results": <array of extraction results>,
    "pretext": <message string elaborated by the
       model>,
    "summary": <summary string of results>,
    "chart": <object for Vega-lite-Altair chart
       specification>,
    "cached": <boolean for cached results>,
    "jumps": <array of associative objects of entry
       table-column>
}
```

B.2. Python Adapter interfaces

```
{
    "name": "Client plug",
    "parameters": [
        {
            "name": "action",
            "type": "string"
        },
        {
            "name": "<argv>",
            "type": "string|number"
        }
```

```
    ]
}

{
    "name": "Adapter",
    "parameters": [
        {
            "name": "action",
            "type": "string"
        },
        {
            "name": "<argv>",
            "type": "string|number"
        }
    ]
}

{
    "name": "Schema tagging",
    "parameters": [
        {
            "name": "dbConnector",
            "type": "mysql.connector"
        }
    ]
}

{
    "name": "Schema fetching",
    "parameters": [
        {
            "name": "dbConnector",
            "type": "mysql.connector"
        },
        {
            "name": "needle",
            "type": "string",
```

```
            "optional": true
        }
    ]
}

{
    "name": "Data visualization",
    "parameters": [
        {
            "name": "dbConnector",
            "type": "mysql.connector"
        },
        {
            "name": "prompt",
            "type": "string"
        },
        {
            "name": "sessionGUID",
            "type": "long"
        }
    ]
}

{
    "name": "Data summarization",
    "parameters": [
        {
            "name": "dbConnector",
            "type": "mysql.connector"
        },
        {
            "name": "keywords",
            "type": "string[]"
        },
        {
            "name": "sessionGUID",
            "type": "long"
```

```
86          }
87      ]
88  }
89
90  {
91      "name": "Data creation",
92      "parameters": [
93          {
94              "name": "dbConnector",
95              "type": "mysql.connector"
96          },
97          {
98              "name": "dbId",
99              "type": "number"
100         },
101         {
102             "name": "prompt",
103             "type": "string"
104         }
105     ]
106 }
107
108 {
109     "name": "Navigation",
110     "parameters": [
111         {
112             "name": "dbConnector",
113             "type": "mysql.connector"
114         },
115         {
116             "name": "dbId",
117             "type": "number"
118         },
119         {
120             "name": "sessionGUID",
121             "type": "long"
122         },
```

```
123         {
124             "name": "connections",
125             "type": "string[]"
126         }
127     ]
128 }
129 
130 {
131     "name": "Intent matching",
132     "parameters": [
133         {
134             "name": "dbConnector",
135             "type": "mysql.connector"
136         },
137         {
138             "name": "prompt",
139             "type": "string"
140         }
141     ]
142 }
143 
144 {
145     "name": "Dashboard export",
146     "parameters": [
147         {
148             "name": "dbConnector",
149             "type": "mysql.connector"
150         },
151         {
152             "name": "queries",
153             "type": "string[]"
154         },
155         {
156             "name": "titles",
157             "type": "string[]"
158         }
159     ]
```

```
}

{
    "name": "Insert intent",
    "parameters": [
        {
            "name": "dbConnector",
            "type": "mysql.connector"
        },
        {
            "name": "dbId",
            "type": "number"
        },
        {
            "name": "prompt",
            "type": "string"
        }
    ]
}

{
    "name": "Select intent",
    "parameters": [
        {
            "name": "dbConnector",
            "type": "mysql.connector"
        },
        {
            "name": "dbId",
            "type": "number"
        },
        {
            "name": "prompt",
            "type": "string"
        },
        {
            "name": "sessionGUID",
```

```
197                 "type": "long"
198             },
199             {
200                 "name": "step",
201                 "type": "number",
202                 "optional": true
203             }
204         ]
205 }
```

List of Figures

List of Tables