Transforming Data Flow: Generative AI in ETL Pipeline Automatization

Master Degree Thesis in Data Science and Engineering

Academic Advisor:
Prof. Daniele Apiletti
Company Advisor:
Dr. Leonardo Contini

Candidate:
Chiara Van der Putten

April 2024

Abstract

In the evolving landscape of enterprise data management, automating the creation of ETL pipelines emerges as a crucial objective. This master's thesis delves into employing state-of-the-art Artificial Intelligence techniques to streamline the integration and transformation of enterprise data, aiming to minimize the manual effort in developing data processing workflows.

In partnership with Mediamente Consulting Srl, the study focuses on designing and implementing a system that efficiently addresses user requests within the ETL framework, leveraging cutting-edge technology.

To this end, a tailored algorithm was designed to process user requests, employing sophisticated data representation techniques to encapsulate the semantic nuances and contextual cues embedded in these queries. This distributed representation of user requests serves as the basis for identifying the most suitable ETL solution from a repertoire of available options. Subsequently, the identified solution is refined through a generative model, which further aligns it with the original user specification, thereby improving the congruence and relevance of the final result. In the formulation of the proposed pipeline, a selected set of embedding techniques and generative models were evaluated and tested, culminating in the identification of the most efficient methodologies that could provide answers most attuned to user needs, as clarified in the thesis.

This approach results in an initial ETL solution closely aligned with the user's needs, substantially reducing the manual work usually associated with creating ETL workflows, although not eliminating it.

Keywords: ETL Workflow, Word Embedding, Generative Model

Contents

Summary

Thesis Objectives

In the modern era of data, effective management and integration of information flows are crucial components for the success and competitiveness of businesses. In the Mediamente Consulting company scenario, which operates in an environment characterized by a growing volume of data and operational complexity, optimizing data integration processes plays a fundamental role.

It is within this context that the present thesis is situated, focused on the design and implementation of a pipeline for the automated creation of workflows in SQL Server Integration Services (SSIS). Through the development of this automated framework, the thesis aims to significantly diminish the manual effort typically associated with the creation of data workflows, thereby enabling employees to focus on more strategic tasks and initiatives within the organization.

The automation project is divided into several stages. At first, custom functions were developed to generate SSIS components. Subsequently, two databases were built: one to collect users' automation needs and the other containing the corresponding modules for generating ETL workflows. After that, a question-answering system was developed and used, which integrates an embedding-based model to identify the most relevant query among those collected and a generative artificial intelligence model to process and customize the answer according to the user's specific needs.

The Proof of Concept (PoC) of this thesis aims to demonstrate the effectiveness of integrating syntactic embedding methodologies and advanced AI technology to automate ETL processes from user requests formulated in natural language.

Research Area and Contribution

During this study, several technologies used in the development of the algorithm for generating the SQL Server Integration Services (SSIS) flow were examined, tested, and evaluated.

As an initial phase, a comprehensive review was conducted to develop custom executable functions for creating SSIS components in XML format, responsible for processing data and defining workflows. It is important to point out that no Python libraries are currently available for generating the XML of these components, which makes this development phase essential.
These functions are employed to specifically address user requests through the development of two separate databases. The first collects the main automation needs identified among employees, formulated in the form of questions, while the second lists the related functions, or combinations thereof, needed to generate the required components or sequences of operations.

Utilizing these databases, a question-answering system has been developed that, employing the technique of embedding, enables the identification of the most relevant need from the collection based on the query formulated by the user. To enrich the effectiveness of embedding, a semantic weighting technique was adopted, which played a crucial role in increasing the accuracy of the model.
Given the importance of an effective vector representation that accurately captures the contextual meaning of the questions, the selection of the word embedding model was made considering key factors such as database size, data features, and application purposes.

After the system has identified and retrieved the most congruent response to the user's request, it is fed into an AI generative model to more closely match the identified function to the user's request, allowing for a more precise and customized solution.
Similar to the evaluation of text embedding models, several options were examined for the LLM, taking into consideration the performance, cost, and efficiency of the various models.
An analysis of different prompt engineering techniques was conducted to evaluate which among them offered optimal results, evaluating them both individually and in combination. During this evaluation, the cost and effectiveness of each technique were examined.

Result and Future Works

The results obtained from the developed pipeline demonstrated that the synergy between text embedding models and Large Language Models (LLMs) can be exploited to tailor the most similar response identified, to the specific needs of the user. This capability stems from the ability of the generative model to accurately understand, through an optimal prompt, which specific function parameters need to be modified to customize the response.

By expanding the database with a wider range of queries and increasing the number of functions to represent more SSIS elements, the range of manageable queries could be greatly expanded. That database expansion would greatly improve the results obtained, paving the way for a refinement of the model's understanding. This would allow the model to interpret and respond autonomously to user queries, without depending on particularly elaborate prompts or examples to guide it through the task.

In conclusion, the pipeline developed proved effective in mitigating the workload required to create the data management flow from scratch, representing an effective solution for improving the overall efficiency of the process.

Introduction

The creative capabilities of Artificial Intelligence have heralded new vistas across domains like computer vision, natural language processing, and the arts, revolutionizing these fields. In the realm of Artificial Intelligence, the usage of generative models has seen widespread adoption, leading to fresh perspectives and innovations across various sectors.

This study focuses on applying these models to automatically generate XML files, which is pivotal in constructing ETL pipelines. It delves into how integrating these models can offer an innovative alternative, significantly reducing the manual labor typically required for crafting XML files crucial for data mapping.

This need emerged within Mediamente Consulting, a company that operates in the field of consulting and management design of decision-support systems. Offering a broad spectrum of services, including data integration and management, data visualization, technology infrastructure, business intelligence, and enterprise performance management (CPM), Mediamente Consulting serves as a strategic partner for corporate clients seeking advanced solutions in these areas.

The thesis is divided into five chapters: the first three are devoted to a detailed analysis of management systems, data collection, and the evolution of the most advanced generative technologies; the last two chapters, on the other hand, focus on the practical implementation of the model, describing the tests carried out, the results achieved and their limitations.

Chapter 1 provided an in-depth analysis of the major enterprise information management systems, highlighting their purposes, advantages, disadvantages, and distinguishing features. Next, emphasis was placed on the architecture of data warehouses, essential pillars for the structured analysis and arrangement of data within organizations.
These systems are crucial to the ultimate goal of ETL (Extract, Transform, Load)

processes, which is to organize data, facilitating effective and efficient information management.

In Chapter 2, the features of ETL flows were explored in detail by examining the standard framework adopted by Mediamente for data management. This analysis included an exploration of the various stages of the workflow, with a focus on the specific objectives, rules established, and changes applied to the data at each step. In addition, a comparison of the main tools used by the company to create ETL flows was conducted. Among these tools we find SSIS playing a central role in this study, as the goal was to develop a method to automatically generate XML files interpreted by this tool starting from user requests expressed in natural language.

In Chapter 3, the analysis focused on the fundamental elements of the project, beginning with an exploration of word embedding models, this section deliberates in detail on their distinctive structure, the functions they perform, and the limitations they encounter.
The discussion then evolved to more elaborate systems, such as Recurrent Neural Networks (RNNs), focusing on the architecture capable of recognizing the sequential relationships present in texts and addressing the main challenges associated with these models, such as the Long Short-Term Memory (LSTM) problem and the strategies adopted to overcome it.
The treatment then follows with an in-depth look at Transformer models, minutely analyzing their structure, which is distinguished by the presence of two key components, the Encoder, and the Decoder, as well as the attention mechanisms that constitute their main innovation.

The focus then turned to question-answering systems, recognized as crucial elements for the developed application and in a wide range of areas. The starting point was the standard architecture of these systems, which are designed to process questions and generate answers, paying particular attention to generative question-answering systems.
Next came an in-depth analysis of the evolution of Self-Supervised Learning (SSL) in the field of Natural Language Processing (NLP), starting with Pretrained Language Models (PLMs). The latter represented a paradigm shift in NLP, introducing the concept of SSL and transfer learning, different pretrained models such as BERT, GPT-1, and GPT-2 were examined, highlighting the peculiarities of each and showcasing their respective strengths and weaknesses.
In conclusion, the chapter focused on Large Language Models (LLMs), with a special interest in models such as GPT-3 and GPT-4, which will be tested in the

pipeline. This section highlighted the innovative techniques and paradigms introduced by these models in the artificial intelligence landscape.

After exploring more theoretical chapters, the focus shifted to concrete implementation details: an in-depth analysis was conducted to clearly illustrate to the reader the structure in XML of the different components used in SSIS for data processing. This step is intended to provide a deeper understanding of both the development of the project and the importance of each element within the SSIS tool.
Next, the structure of the functions representing the components was outlined: the execution of these functions generates the XML code. The Excel document structure essential for extracting function parameters and describing the configuration of the tables from which data is extracted for flow operations was also described.

The following section explores the pipeline structure in detail, beginning with the tokenizer employed to generate word embeddings for both database queries and the user-posed question, along with the distance metric adopted to identify the question in the database most akin to the user's question. Next, there is an in-depth analysis of the various prompting techniques tested, including concrete examples of their application.

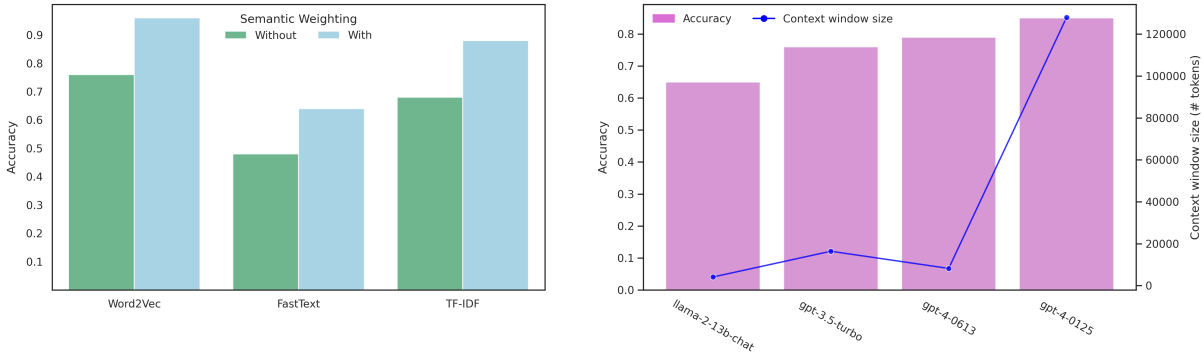

Figure 1: Accuracy comparison between embedding models with and without semantic weighting (left), accuracy and context windows comparison between LLMs (right)

Chapter 4 culminates in a detailed illustration of the experiments conducted to refine the final pipeline and analyze the results achieved. This research phase involved evaluating various embedding models to identify the best performance, following a similar process for generative models to identify the most appropriate one for the specific application task.

Results from tests conducted on the different embedding models, both with the application of the semantic weighting technique and without, as well as on the various language models, are illustrated in Figure 1. The introduction of semantic weighting led to a significant increase in the performance of all three methodologies examined. Among the Large Language Models (LLMs) analyzed, the model that demonstrated the greatest accuracy is the latest one released by OpenAI, which is also characterized by the largest context window.

In addition, various prompting strategies were explored, and the results achieved in terms of accuracy and the average number of tokens needed per request with the different prompting techniques are shown in Figure 2.

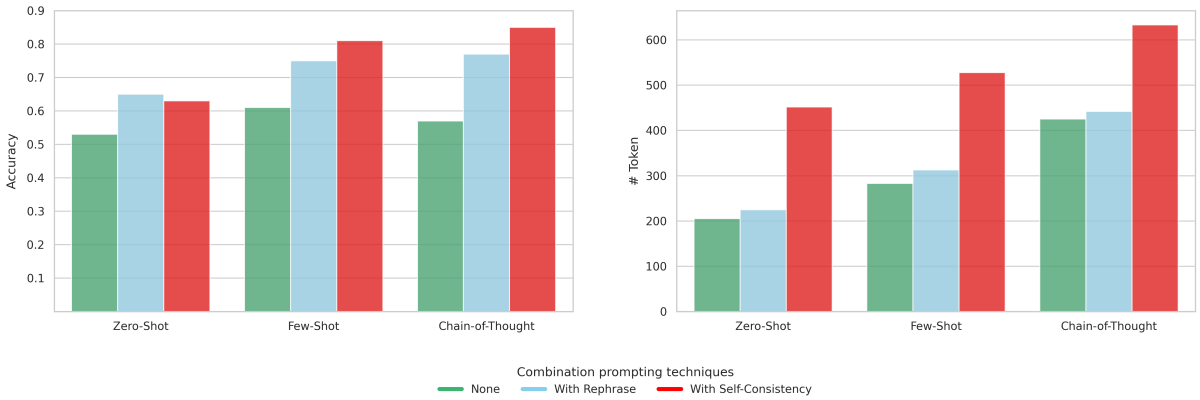

Figure 2: Comparison of the accuracy (left) and average number of tokens per request (right) of various prompting techniques, both individually and in combination with Self-Consistency or Rephrase and Respond techniques

Substantial improvement in results can be observed when a combined approach of various prompting techniques is adopted. By integrating methods specific to the domain of reasoning and logic, techniques designed to handle new tasks without the need for extensive training, and strategies aimed at understanding the user's intentions, a significant increase in the quality of performance is achieved. This synergy between different methodologies not only refines the accuracy of responses but also amplifies the system's ability to adapt effectively to a wide range of new tasks.

Fine-tuning was attempted, which, however, did not achieve the desired results due to the limited amount of examples available in the database. The chapter also

presents the pseudo-code of the pipeline, providing a comprehensive view of the development process.

The thesis concludes with a final chapter that examines in detail the limitations of the survey conducted, the results achieved, and outlines prospects for future developments. The latter particularly focuses on the importance of database expansion, identified as a crucial element in improving and refining research outcomes.

Chapter 1

Data Warehouse

A Data Warehouse is an aggregate of structured historical data that comes from different sources, derived from business activity, taken from external sources, generated by the applications, log files and more. The data is first transformed from heterogeneous data into congruent information that will be accessed by responsible decisions through Business Intelligence tools or other analytics applications.

The characteristics that distinguish DW from other decision-making tools have been outlined by William H. Inmon [13] and include:

- **Subject-Oriented**: the data in DW are organized by relevant subjects in order to offer all the information related to a specific field or area
- **Integrated**: Data from different sources often show heterogeneity in terms of format and encoding and are then integrated and consolidated within the DW
- **Time-Variant**: the DW contains data belonging to a wider time horizon than operational systems, this allows the analysis of trends and the evolution of information over time
- **Non-volatile**: once the data has been uploaded within the DW become immutable, they do not undergo updates or cancellations while maintaining their integrity over time

1.1 Data Lake, Data Mart, Database

It is crucial at this point to distinguish between the different systems of management of company information resources. In this context, let's now define the

peculiarities, needs and objectives for which Data Lake, Data Mart, Data Warehouse and Database are designated.

Data Lakes are large scalable repositories that contain great amounts of data in their original and raw form. The raw nature of the acquired data requires the use of more advanced and dynamic analysis technologies than those typical of a Data Warehouse. However, using a Data Lake reduces data ingestion costs by providing a comprehensive view of enterprise information resources, simplifying big data analysis and retaining flexibility for subsequent processing [20].
Data lakes represent a valuable resource in various sectors: in finance, for example, their ability to store and update market data in real time is crucial to making timely decisions, in the field of IoT, where sensors generate huge amounts of semi-structured data on the surrounding environment, streaming platforms, which collect real-time data on customer habits, find in data lakes an essential resource to power recommendation algorithms, Figure 1.1.

As can be seen from the previous examples, the main purpose of a Data Lake is to collect data immediately to decide how to use it only at a later time, unlike DW where storage can take months or even years. The data that will then be used for analysis are processed only at a later time, this type of scheme is called "on read".

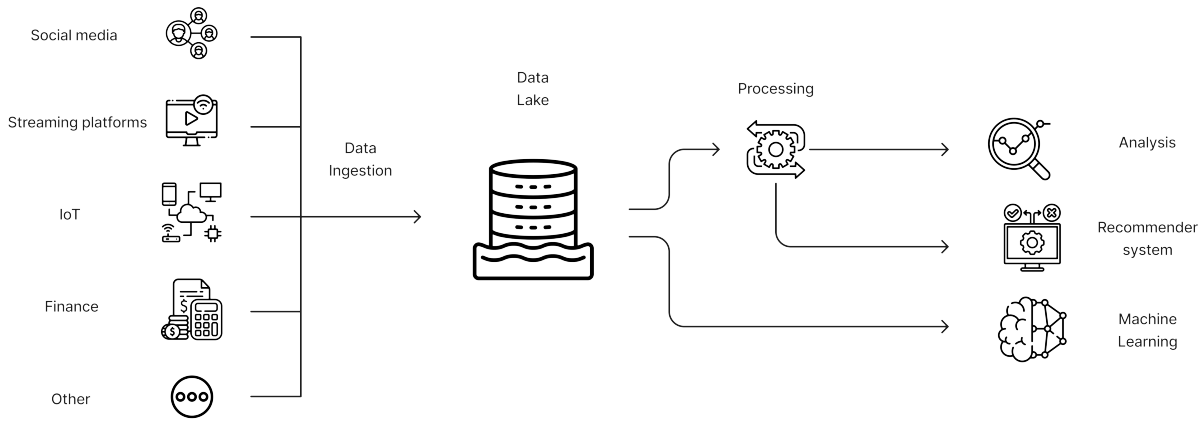

Figure 1.1: Organization and operation of a Data Lake system

Data Marts, on the other hand, have a similar structure to that of Data Warehouses, because they contain consolidated and consistent data, ready for analysis. However, their specificity lies in the optimization of the needs of a particular group

or business process. Essentially, they are configured as single-issue data warehouses that reflect the needs of a specific business unit, such as marketing [11].
The advantages of Data Marts are many: first, you have a remarkable speed in processing queries, thanks to the reduced amount of data managed, in addition, they offer a high level of security, limiting the visibility of data to individual departments and allowing targeted security control. The reduction of complexity is an additional benefit that translates into a significant simplification in management and maintenance since the data are limited to a single functional scope.
Despite these indisputable advantages, Data Marts have some limitations. Their vision focused on a single department may be limited compared to the Data Warehouse, which offers a wider overview of business data. In addition, there is the possibility of data duplication, as several Data Marts may share similar information.

The term *Database* refers to any organized collection of data used for storage, accessibility, and recovery. Typically, databases are circumscribed to a single application and are optimized for fast read and write operations. While commonly referring to online databases for transactional processing, it is important to note that they can take different forms, including formats such as XML, CSV files, or Excel spreadsheets.
The main differences concerning Data Warehouses emerge both from the functional point of view, as the databases are designed to meet operational needs, and both from the design point of view, as they are oriented to manage daily operational activities, reflecting the dynamics and volatility of the data.
The categories of users accessing the two systems are also significantly different, as databases are queried by a wide spectrum of users belonging to different categories.
A further element of distinction lies in the backup strategy. Since database data is subject to frequent changes, they require repeated saves to ensure the integrity of information over time [36].

The interaction between Database, Data Warehouse, and Data Mart is illustrated by Figure 1.2. Each transactional database represents a separate source, from which data and other sources are extracted and processed through an ETL flow before being uploaded to the DW. From the latter, several data sets related to specific topics are loaded into the various data marts, which together with the Data Warehouse will be used to generate a variety of reports and dashboards.

While entity-relationship diagrams and normalization techniques are used in the OLTP environment to construct transactional databases, this is improper for the

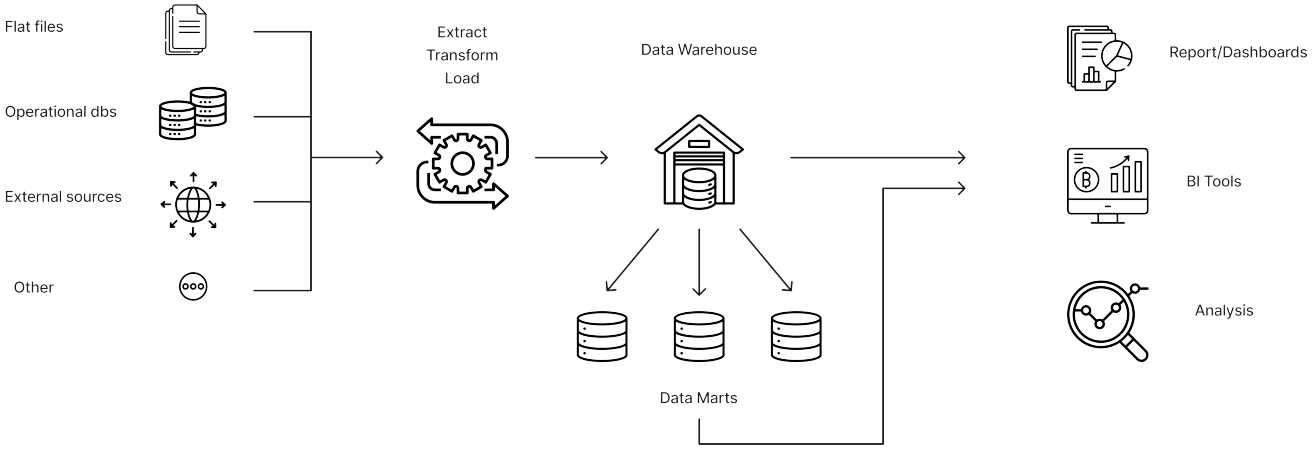

Figure 1.2: Organization and operation of a Data Warehouse system in the field of Business Intelligence

decision support system design. The most common types of diagrams for DW construction will now be explained.

1.2 Data Warehousing Architectures

1.2.1 Star schema

A typical implementation for Data Warehouse construction follows the star diagram model. As the name suggests, this scheme is similar to a star, characterized by a large central table and a series of smaller tables placed around it. An example of this provision is clearly illustrated in Figure 1.3.
The configuration of the star schema is asymmetric: the main table, the table of facts, placed in the center of the diagram, is the only one equipped with multiple links that are concretized through the keys connecting with the tables of dimensions. The fact table preserves the numerical measures of the business, each of which represents the intersection of all the dimensions.

The *Primary Key*, crucial for uniquely identifying individual records in the facts table, is crafted by combining *Foreign Keys* from the dimension tables. Nevertheless, owing to the complexity of an FK, which may involve multiple fields or lengthy attributes like the fiscal code, the introduction of the *Surrogate Key* becomes pivotal. Surrogate Keys, represented by incremental numeric fields, assume the role of the PK for dimension tables, their values lack intrinsic significance with

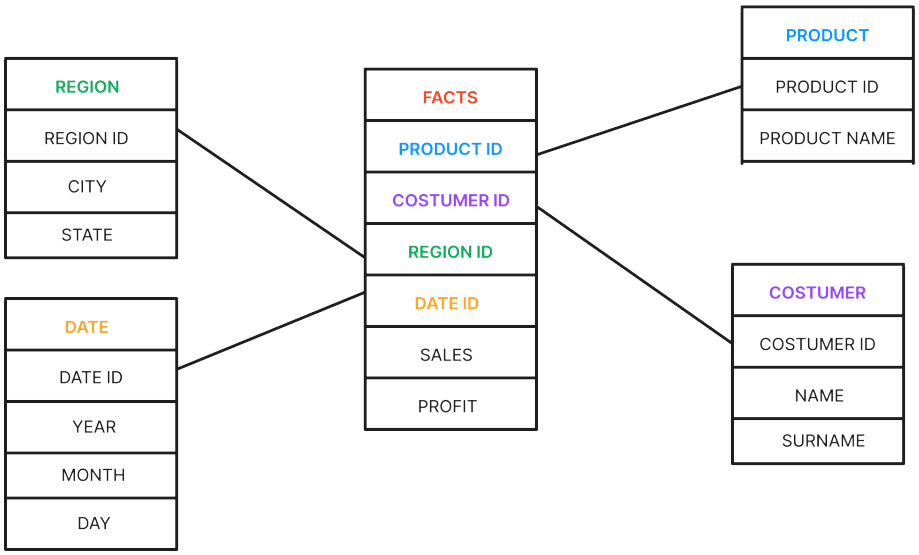

Figure 1.3: Star scheme example

the data. Embracing SKs simplifies the administration of relationships between tables and enhances the efficiency of search and join operations.

In the star scheme, the facts table is normalized, including a single key combination of the dimensional elements, while the dimensional table is denormalized. This approach reduces the number of join operations but increases the required memory size. This model is particularly advantageous in scenarios where the commercial entity to be represented includes a large number of records, for example, could be adopted by a commercial company that wants to monitor the movements of products in the warehouse [21].

1.2.2 Snowflake schema

The snowflake schema has a similar structure to the star one, with the key distinction that some of the dimensions are normalized. This means that these dimensions are divided into smaller, related tables, this subdivision aims to reduce data redundancy. For example, a hierarchical structure could be implemented, where a dimension such as "Date" is fragmented into separate tables for "Year" and "Month", while maintaining a hierarchical relationship between them, Figure 1.4.

The main objective of the snowflake scheme is to save space by minimizing data redundancy. However, it should be noted that this additional normalization can lead to greater complexity in queries. Queries must cross a larger number of joins between normalized tables, increasing the complexity of querying operations.

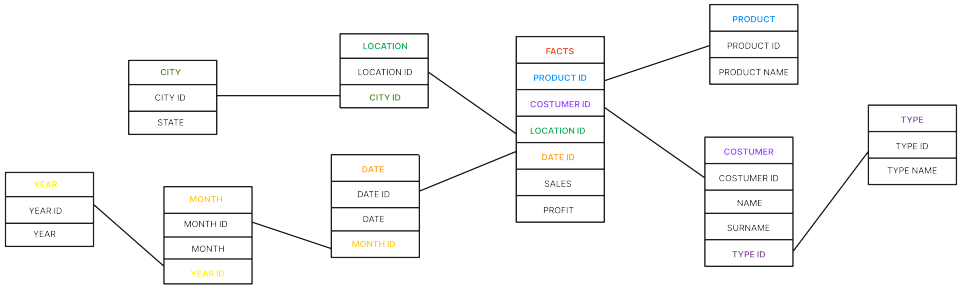

Figure 1.4: Snowflake scheme example

1.3 OLAP and OLTP

After identifying the most suitable structure for data storage, the next step is to identify the most suitable data processing system. Currently, the available systems are divided into two main categories: *On-Line Transaction Processing* (OLTP) and *On-Line Analytical Processing* (OLAP).

Transaction management systems support operations such as financial transactions, order processing, or store upgrades, generally responding to the company's daily operational needs. These systems adopt an entity-relationship data structure to ensure data integrity.
Since the focus is on transactional data, temporary imperfections in the data, such as incomplete or ongoing transactions, are tolerated. An example of this could be the lack of information about a user's tariff plan during a change of plan, which does not compromise the overall integrity of the data. In contrast, the presence of erroneous tariff data would pose a significant issue.

On the other hand, analysis management systems adopt multidimensional data models, and data warehouses, to analyze information from different perspectives. As these systems are used for analytics and to support business decisions, they do not tolerate "dirty" data that could compromise information processing, such as the presence of null fields. Therefore, data cleansing operations are performed.

Unlike OLTP systems, inaccuracies in the data do not affect analysis, as they are hidden by aggregation operations across the entire dataset [21].

As highlighted by the distinct organizational structure of the data used by the two systems, the operations performed on them exhibit significant differences. In OLTP systems, the emphasis is primarily on data writing operations, such as updates, inserts, and deletions, along with daily operational needs. Consequently, the environment is optimized to efficiently support such transactional operations. In contrast, OLAP systems primarily revolve around analysis and involve complex ad hoc queries aimed at analyzing consolidated data with a broad temporal horizon [4].

The distinction between users of the two environments is based on their respective roles. OLTP tools are intended for use by employees, customers, and IT professionals, while OLAP tools are designed for specialized workers such as managers, analysts, and data scientists [27].
The main differences between the two systems are shown in the Table 1.1.

Features	**OLTP**	**OLAP**
Function	daily management	decision support
Accuracy	detailed data	consolidated data
Data structure	relational models	multidimensional models
Data structure	real-time or every transition	planned, usually daily
Time window	limited	historic
Access type	read/write	read
Query structure	simple and repetitive	complex and ad-hoc

Table 1.1: Key Differences: OLAP vs. OLTP

Chapter 2

Foundations of ETL Pipelines

Due to the exponential surge in data, businesses are grappling with a vast volume of information stemming from various sources: SQL databases, NoSQL databases, standard text or XML files, cloud platforms, and more. Interpreting these diverse data sources is challenging due to their disparate and disorderly formats. As a result, it is becoming more important to process and standardize this data to make it uniform and readily available for analysis.

The ETL process involves three basic steps:

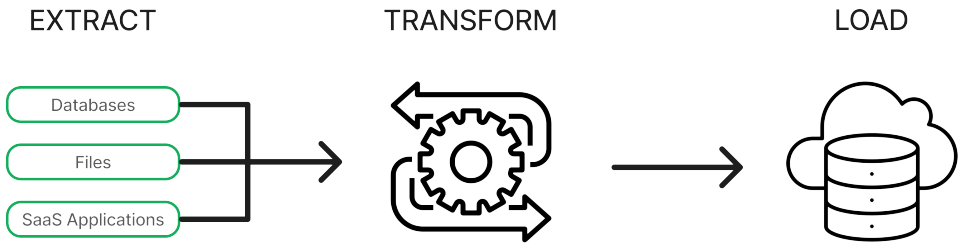

Figure 2.1: The basic ETL process used in Data Warehouses

- **Data Extraction:** The goal of this phase is to gather data from a variety of sources, with complexity differing depending on the source of the data. There are two ways to extract data: static ingestion involves gathering all the available records from a source and transferring them to the next steps of the ETL process and incremental ingestion in which only new, updated, or newly deleted records are acquired through each iteration.

- **Data Transformation:** This phase is designed to convert data from different organizational forms and formats into a uniform format. Dealing with some redundant, ambiguous, incomplete, and anti-rules, data to form a unity of data granularity and data format. [32]
- **Data Loading:** This final phase is responsible for loading the data processed by the above two steps into the Data Warehouse. There are two main approaches to loading data: the refreshing method is mainly used to load the data into the database while creating a Data Warehouse, while the updating method is used to maintain the data warehouse. [32]

This process ensures data consistency, making it suitable for analysis, and overcomes its initial heterogeneity and disorderliness.
The challenge in standardizing ETL processes lies not so much in the data loading and extraction phases, which follow established models such as entity-relationship diagrams or star and snowflake schemes. The true challenge comes during the data transformation phase, which varies depending on the specific needs of the data or analyses involved [30].

2.1 Data Integration ETL Framework

A data pipeline is a series of operations that involve manipulating and transforming data through a series of components. By processing the output of the previous operation and serving as input for the subsequent one, each component establishes an automated flow in data management [34]. Typically, this structure begins by receiving incoming data and ends with a data sink that stores post-processed data within the Data Warehouse.

Subsequently, an extensive analysis will be carried out on the implemented data flow framework within the company, analyzing its operational specifics and features.

2.1.1 Level L0

The initial step, as depicted in the Figure 2.2, involves **Staging**, which essentially encompasses transferring the source data into staging tables as-is, without conducting any data validation or transformation. The source systems providing this data can vary, with the most common being operational systems within databases or files generated by suppliers.

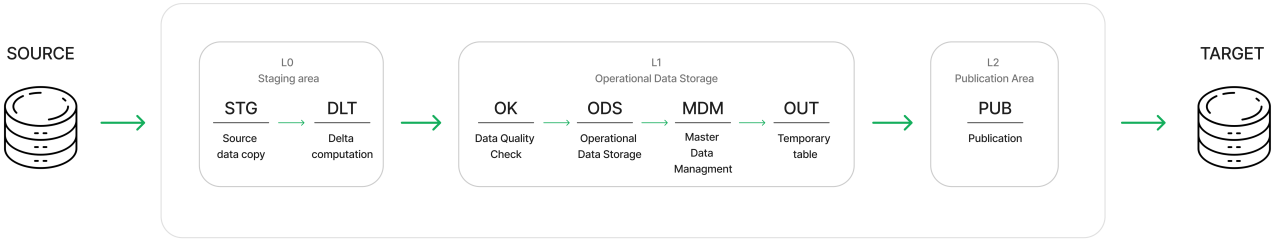

Figure 2.2: Structure of company Data Flow

Furthermore, during this phase, a variable named `JOBID` is appended to each record. This variable usually consists of a twelve-digit code that represents the year, month, day, hour, minutes, and seconds, thereby indicating the data's association with a specific loading stream. This numerical choice is aimed at optimizing computational performance for future filtering operations.

As loading into the staging tables occurs regularly, it becomes essential to identify changes in the current loading stream data compared to the previous one. This operation, called **Delta**, can be executed in two modes:

- If within the metadata tables there is information about the latest data update, new added data is retrieved, while a log table tracks deleted records.
- In the absence of update date information, two 'minus' operations are performed: one between today's and yesterday's data to identify added records, and another between yesterday's and today's data to determine the deleted data. In both cases, a variable named `FLG_NEG` is added, taking values 0 or 1 to denote whether the record was added or deleted, respectively.

2.1.2 Level L1

The most significant and expensive operations of the entire ETL process take place in the second stage of the flow. The purpose of this phase is to transition from replicating source structures to integrating and reshaping information, which is then transformed into tables that are tailored for the next tier's analysis.
Transformations at this stage include:

- Normalization
- Integration

- Generation and resolution of internal keys

The **Data quality check** phase is central because the presence of low-quality data could result in a range of issues within the Data Warehouse, such as failed loading due to constraint violations or inaccurate data types, as well as the inconsistencies that necessitate the application of business rules [28]. At first, the process selects solely unique records with higher `JOBID` values, which encapsulate the most recent data snapshots from the delta tables. Following this, these data undergo a comprehensive validation process involving various rules:

- **Referential integrity rules**: are constructed to ensure that data consistency is maintained across various tables in a database. These rules establish constraints that must be met to ensure valid relationships between tables, ensuring that data is correctly associated and consistent within the system [16]. This check involves joining different tables in the database using their respective foreign keys.

- **Record validation rules**: The purpose of these checks is to discard records that do not meet the predefined criteria. Not null constraints are enforced in fields where NULL values are not permissible, and this is usually resolved by replacing NULL values with defaults [16]. To ensure data integrity, attribute-based check constraints, like length restrictions and invalid value identification, are utilized [28].

- **Business rules**: are criteria set by users to identify suspicious values through comparisons or statistical methods. They encompass missing values, data errors based on specific business definitions, and outliers [28].

After validation through these various rules, the data is loaded into the OK tables.

In the **Operational Data Store** layer, consolidated and historicized data from the OK tables is loaded. The aim is to have a replica of the source after performing reference and data quality checks in previous steps. To update data from various flows, a constraint on the primary key is utilized. This allows for specific handling of records: if the record already exists, it will be discarded; if it exists but with different values in secondary fields, the record will be updated; otherwise, it will be inserted into the ODS tables.

During the **Master Data Management** phase, two essential operations are carried out: loading data from various sources and enriching it.

The first operation is carried out by joining the ODS fact tables to form a single central table, which is used to combine data from multiple sources. This type of table joining is fundamental as the steps seen earlier are carried out separately for each table. Additionally, this step allows prioritization between sources. For instance, if there are two tables—one from an API and the other from an ERP—employing a left outer join can prioritize the table located on the left side of the join.
Data enrichment is achieved through join operations, utilizing dimensional tables that offer additional descriptive details. This approach enriches core data with supplementary information.

The implementation of surrogate keys (SK) to replace natural keys within dimensional tables is a crucial process within MDM. These keys are designed to enhance the efficiency of table joins by reducing computational load and speeding up operations. Surrogate keys encompass a meaningless incrementing integer value, simplifying inter-table relationships [37].

At the final stage of Level L1, known as **OUT**, the enriched and integrated data from prior operations is readied for publication. The OUT tables, temporary in nature, serve as a snapshot of the publication. This phase functions as a preparatory step for the data model at the subsequent Level L2. Here, the importance of surrogate keys becomes evident, employed in the various joins that contribute to constructing the final chosen Data Warehouse model.

2.1.3 Level L2

Level L2, which is the final level, only requires one step: **Publishing**. This level's tables contain data divided into thematic areas, usually aligned with specific business processes. The tables will be divided into:

- *Fact tables* is where the numerical measurements of the business processes are stored. These measurements or events are related to each dimension table by foreign keys.
- *Dimension tables* contains attributes used to constrain, group, or browse the fact data [29].

The Data Warehouse structure is made up of these tables that can be configured with:

- *Star schema* is a representation of multidimensional data that is made up of a fact table and a set of unnormalized dimension tables using the relational model.
- *Snowflake schema* is obtained from a star schema. It consists of a fact table and several partially normalized dimension tables [10].

Both models have different advantages and disadvantages, as shown in the Table 2.1. The choice of the type of scheme depends on the specific needs of the company and the type of analysis that will be carried out on the data. In general, the preferred model is the one that minimizes the number of joins required for analyses, as these operations are computationally expensive.

In this final stage, the complete set of business data intended for analysis and decision-making is collected. Here is where it is possible to create data marts and data collectors targeted to specific analytical interests. To perform an accurate analysis, the most recent images from the OUT phase must be selected from all available images, performing the same procedure as in the delta phase.

	Star schema	**Snowflake schema**
Integrity	Lower, due to denormalization	Higher
Data redundancy	Higher	Lower due to normalization
Complexity	Simplified model and fast queries	Complex model and queries
Execution time	Higher due to redundant data	Lower due to normalization
Maintenance	Easier due to the reduced number of relationships	Harder
OLAP friendly	Yes	No

Table 2.1: Comparison between the star and snowflake schemes

2.2 ODI vs SSIS

Key tools used by the company to create ETL streams include Oracle Data Integrator and SQL Server Integration Services.

ODI is a highly used, comprehensive platform that covers all of the data integration requirements, starting from high-volume, high-performance batch loads, to event-driven, trickle-feed integration processes, to SOA-enabled data services [22]. A key distinction of ODIs compared to other ETL tools is its architecture based on Extract, Load, Transform (ELT). This approach eliminates the need for an ETL

server between the source data and the destination server. In practice, it allows the extraction of data, their direct upload to the destination server, and then the execution of transformations on the data within the server itself.

SSIS is a powerful platform for creating comprehensive solutions that cater to data integration and transformation needs within an enterprise. This tool tackles complex business challenges by extracting and transforming data from diverse sources, such as XML files, flat files, and relational databases. Once the processing is complete, the data is loaded into one or more destinations as needed [31].

ODI leverages an external tool to automate the generation of the entire configuration of the data flow, aligning with the specific requirements and structures defined in an Excel sheet outlining the flow structure. This functionality enhances the efficiency of the ETL workflow, considerably streamlining the development time required for data integration solutions. Recognizing the absence of a similar feature in SSIS, the thesis focuses on the creation of an implementation aimed at automating the process.

Another significant difference is in the approach to data transformation: in ODI, a push-based approach is adopted, which implies that data transformation operations take place directly within the target database, fully exploiting the computational power of the latter [9]. In contrast, in SSIS data is extracted and transformed into a separate environment.
From the perspective of scalability, the Oracle tool stands out positively thanks to its distributed architecture, which allows the execution of transformation operations in parallel on multiple server nodes [9]. In contrast, in SSIS, scalability is constrained by the complexity of server configuration and ETL flow.

After this brief comparative analysis of the instruments, it may not be clear why SSIS is preferred. There are two main reasons: firstly, the customer's use of services and tools included in the Microsoft SQL Server ecosystem. This may result from the need to maintain technological consistency with the Microsoft solutions already adopted. Secondly, the cost of licensing should be considered, which is generally higher for Oracle tools.

Chapter 3

Evolution and Revolution of Large Language Models

3.1 Introduction to Large Language Models

The relentless progression of digital technology has ignited an escalating interest in language models. To fully grasp their essence and potential, it is imperative to embark on a journey back in time, particularly to the 1950s, when the linguist Noam Chomsky introduced the groundbreaking theory of generative grammar. Central to this theory is the radical proposition that the grammar of any language can be delineated through a set of formal rules. These rules uniquely generate sequences that are grammatically valid for the language in question, effectively transforming the concept of grammar into a mathematical model.

Despite the promise of Chomsky's theory, the early experiments encountered some difficulties. These included the difficulty of the model to incorporate context and the computational limitations of the time. Only in the 1980s did the advent of the first machine learning models capable of identifying patterns, facilitated by advances in algorithms such as neural networks.

The dawn of the 21st century marked a significant milestone with the development of more sophisticated models trained through unsupervised learning. This advance paved the way for unprecedented growth in the field of neural networks and deep learning.
Today, the advent of Transformer models has revolutionized our ability to process vast datasets, unlocking the ability to understand the context, structure, and semantics of the human language, marking a monumental leap forward in the field

[25].

At this point, a more precise definition of Large Language Models (LLMs) can be provided: they constitute a subcategory of neural networks dedicated to natural language processing (NLP). The hallmark of these models lies in their considerable size and complex capabilities; they are trained on large text datasets to understand intricate language patterns, semantic contexts, and internal relationships between words.

LLMs are applied across a broad spectrum of domains: from enhancing the precision and flow of automatic translations to improving virtual assistants with deeper insights and more conversational responses. Up to applications of automatic summarization of documents or long texts, through identification of key information. Furthermore, these models are instrumental in generating code automatically, powering question-answer systems, and serving a multitude of other purposes, demonstrating their versatility and relevance in a wide range of fields.

Throughout this chapter, the fundamental techniques and methodologies underlying Large Language Models (LLMs) will be explored, analyzing their evolution from the simple structure of recurrent neural networks (RNNs) to the sophisticated architecture of Transformer models. Through this examination, an attempt will be made to outline the progression of techniques used in LLMs, highlighting the key innovations that have made it possible for them to achieve their remarkable natural language understanding and generation capabilities.

3.1.1 Word Embedding: Starting Point for LLMs

One of the main topics in natural language processing is distributed word representation, a technique that represents words as continuous vectors capable of capturing relationships between words. This approach is rooted in the linguistic theory known as Zellig Harris' "*distributional hypothesis*" which postulates that words that appear in similar contexts tend to have similar or related meanings [12].

Based on this assumption, several neural words embedding models have been developed, including **Word2Vec**, introduced by Tomas Mikolov, which stands out as a pioneering and widely adopted framework.
The algorithm takes as input a text, also known as *corpus*, from which it generates an output consisting of a vector representation of the words in the corpus, as shown in Figure 3.1. The model W2V includes two main architecture variants:

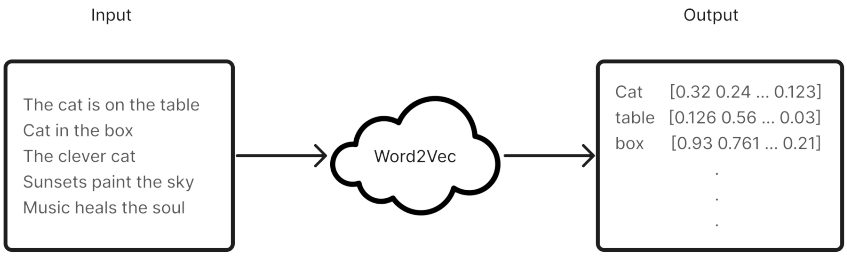

Figure 3.1: Vectorization of the input corpus

- The *Skip-Gram* model, which predicts context words from an input word.
- The *Continuous Bag of Words (CBOW)*, which predicts the input word from context words.

Both variants share the basic goal of learning meaningful embedding vectors for words in a specific context.
The context is defined by a window of words, where its size represents the key parameter of the model; a window of length "t" corresponds to the selection of t tokens before and after the target word.

Both architectures are implemented through a shallow neural network consisting of three layers: an input layer, a hidden layer, and an output layer, Figure 3.2. As easily understood, depending on the model, both the input and output can represent either the window of words or the target word.
The training of embeddings (hidden layer) will also change, in the case of CBOW they will be optimized to maximize the probability of correctly predicting the input word, and in the case of Skip-Gram to maximize the probability of correctly predicting the surrounding words, i.e., the context [19].

Once the word embedding vectors have been computed using one of two architectures, they can be used to compute the similarity between two words, using different measures such as Euclidean distance, Jaccard similarity, or cosine similarity, which among the three represents the most widely used measure.

The cosine similarity formula evaluates the angle between the word embedding vectors in the vector space, indicating how directionally similar the vectors are. A value of 1 indicates that the two words have the highest similarity, while a value of

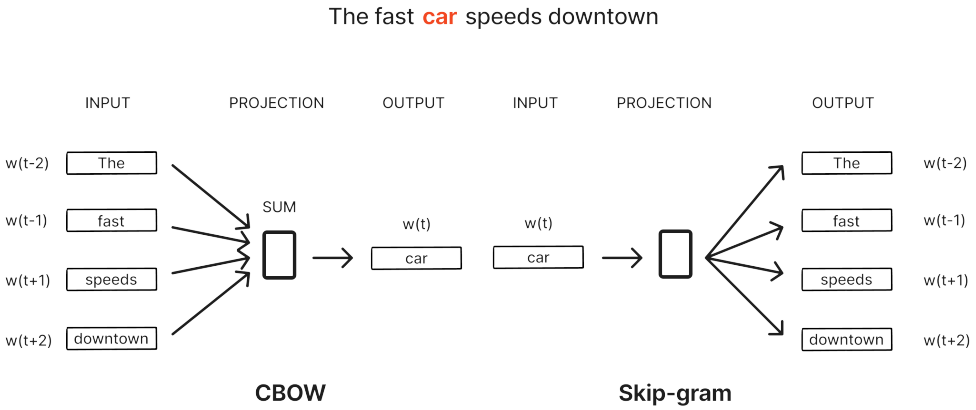

Figure 3.2: Difference between SkipGram and CBOW training architectures.

0 denotes no similarity; conversely, a value of -1 suggests maximum dissimilarity between the words.

$$cos(\mathbf{w_i}, \mathbf{w_j}) = \frac{\mathbf{w_i} \cdot \mathbf{w_j}}{\|\mathbf{w_i}\| \cdot \|\mathbf{w_j}\|}$$

where $\mathbf{w_i}$ and $\mathbf{w_j}$ represent the embeddings of two words and $\|\mathbf{w_i}\|$, $\|\mathbf{w_j}\|$ the respective Euclidean norms of the vectors.

Although Word2Vec is a widely used algorithm that offers excellent performance, it has some notable limitations:

- **Dependence on large amounts of data**: in the absence of a sufficiently large corpus, the model cannot capture the complexity of relationships
- **Poor handling of Out-Of-Vocabulary (OOV) words**: in contexts of a dynamic vocabulary, the absence of words in the vocabulary used in training results in limited flexibility
- **Limitation in context**: because the model considers only a window of surrounding words during training, the broader relationships present in the overall context are not captured
- **Difficulty with homonymous and antonymic words**: difficulty with semantically related but dissimilar words that share similar contexts, such as synonyms and antonyms

To overcome limitations in providing accurate results for rare words and outside the vocabulary, the **FastText** model was introduced, an innovative embedding

technique that builds on the skip-gram concept of the Word2Vec model. Unlike the latter, FastText splits words into smaller substrings, known as *"n-grams"*, allowing the model to capture information about the internal structure of words. Consider the word "eating" as an example. Its representation using $n = 3$ is as follows:

$$< ea, \quad eat, \quad ati, \quad tin, \quad ing, \quad ng >$$

It is evident that $< eat >$ appears among the sequences, corresponding to the word *"eat"*, whose tri-gram, however, is different than that of *"eating"*. In addition, the word itself is also considered within the set of n-grams of the word.

The word embedding vector will be the sum of all the vectors associated with the n-grams. This technique allows for an accurate representation of rare words since some of their n-grams are likely to appear in other words as well [2].
This model has some disadvantages, including longer training times due to the inclusion of n-grams and a lower ability to detect similarities between words than the Word2Vec model.

Another technique used in the context of vector representations is **Sent2Vec**, which extends the fundamentals of Word2Vec to capture the semantics of whole sentences rather than individual words, thus allowing a broader understanding of context to be captured.
The model can be considered an extension of C-BOW, but with some significant differences; for example, it uses dynamic context windows, making it more effective in capturing sentence-level context [24]. However, this greater emphasis on sentence-level context may result in less ability to capture the subtle semantic nuances of individual words.

3.1.2 Technological Advancements in Language Models: From RNN to Transformer

Recurrent neural networks (RNNs) are specific neural network architectures designed to process sequential input, making them key tools in Natural Language Processing (NLP). Their main applications include text generation, machine translation, and sentiment analysis.

In the depicted architecture, Figure 3.3, a recurring pattern of cells forms a sequence, enabling the flow of information from one time step to the next. Each *cell* is fed with an input $x(t)$, such as a word, and the hidden state from the preceding

hidden layer $h(t-1)$, which encapsulates insights from earlier sequence iterations, essentially serving as the network's short-term memory.
Leveraging these inputs, the cell computes the output $o(t)$ corresponding to the current sequence position. Throughout the network's training process, the cell autonomously learns the weights U, V, and W about the three layers by optimizing a loss function [33].

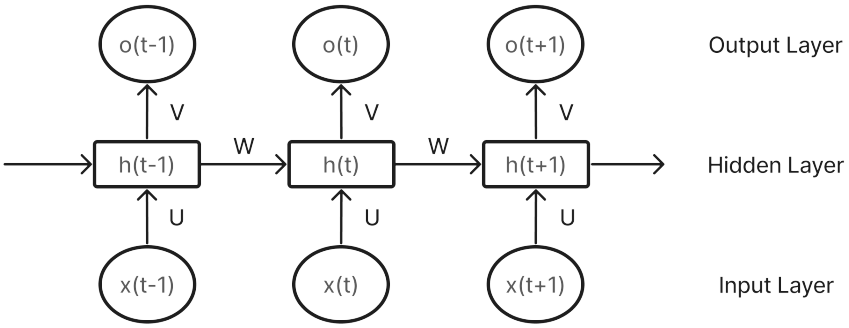

Figure 3.3: Diagram of the Structure of a Recurrent Neural Network

Despite the ability of RNNs to capture sequential dependencies, they suffer from the issue of *vanishing gradients*. In other words, the weight of information from previous inputs decreases exponentially as the sequence progresses, making it difficult to capture long-term relationships accurately [18].

To overcome this challenge, **Long Short-Term Memory** (LSTMs), extensions of RNN architectures but equipped with the ability to effectively handle long-term dependencies in sequences, were developed. This is achieved through the use of a *cell state*, which stores long-term information and allows LSTMs to store and manipulate information more effectively than traditional RNNs.

The architecture resembles that of RNNs, but three gates are introduced within each cell. The first is called the Forget Gate, which, through a sigmoid function, decides what information from $x(t)$ and $h(t-1)$ will be forgotten by the cell state. The second gate is the Input Gate, which operates similarly to the previous gate but determines what new information will be stored in the cell state. Finally, the information obtained passes through the Output Gate to determine what information will be returned, again using a sigmoid function [15].

The limitations of LSTM models focus mainly on the computational complexity

required during both training and inference. In addition, because of their sequential structure, which prevents parallelization across tokens in the input sequence.

Before arriving at modern transformers, the **Seq2Seq** architecture was developed, which consists of two basic components:

- The *Encoder* converts a sequence of input tokens into a series of embedded vectors; it is commonly referred to as the hidden state or context.
- The *Decoder* uses the Encoder's output to generate a sequence of output tokens, one at a time.

In this type of architecture, both components are recurrent neural networks enhanced with an *attention* mechanism [33]. This allows the Decoder to assign a different weight to each token. Specifically, attention is computed using the features of both sequences to determine the importance of each element in the input sequence relative to the elements in the output sequence.

The **Transformer** architecture was originally introduced in the article "Attention Is All You Need." While retaining the encoder-decoder paradigm, the structure of the blocks within the two components is completely revised.

The input is initially tokenized using a Tokenizer, and each token is converted into an embedding vector.
Since Transformers have no inherent sequential structure, it is essential to add *Positional Encoding*. This component adds positional encoding vectors to the embedding vectors, providing the model with information about the position of words in the sequence.
Then input tokens are sent to the **Encoder**, which is composed of a series of decoding layers, each of which includes two attention sublayers:a Multi-Head Self-Attention sublayer and a Feed-Forward sublayer.

We have previously seen what the attention mechanism consists of; the self-attention mechanism differs in that attention is computed among tokens of the same sequence.
This mechanism is implemented through four steps:

- *Creation of query, key and value vectors*: Each embedding of the token x_i is multiplied by three separate weight matrices: one for queries W_Q, one for keys W_K and one for values W_V, learned by the model during the training

process. These operations produce query vectors $q_i = x_i W_Q$, key vectors $k_i = x_i W_K$ and value vectors $v_i = x_i W_V$, respectively.

- *Compute attention scores*: Each query vector is multiplied by each transposed key vector, for each token pair i and j, using the following formula:

$$\text{score}(q_i, k_j) = q_i \cdot k_j^T$$

This process generates a matrix of scalar products indicating how "similar" each token is to the other tokens in the sequence; a sequence of N tokens corresponds to a matrix NxN of attention scores.

- *Calculate attention weights*: To ensure stability in the gradient during the training process, the previously calculated attention scores are divided by the square root of the size of the key vectors, $\sqrt{d_k}$, and then normalized through a softmax function. The resulting matrix $N \times N$ will now contain all the attention weights w_{ij}.

- *Update token embeddings*: After the weights are calculated, they are used to weight the value vectors. For each token i, the output y_i will be the weighted sum of the value vectors using the weights w_{ij}, so $y_i = \sum_j w_{ij} v_j$.

The attention function is executed in parallel h times, each time with a different set of weight matrices (W_K, W_V, W_Q), known as the *attention head*, as shown in Figure 3.4.

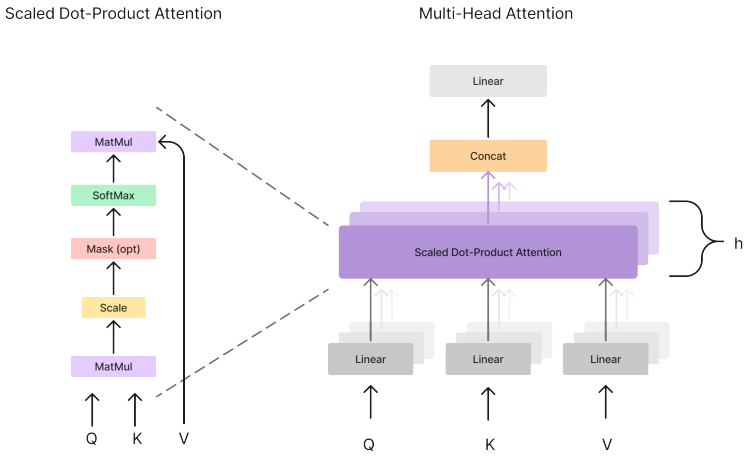

Figure 3.4: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention

Before using these matrices, they are linearly projected independently, these different projections allow the attention function to focus on different semantic aspects. The outputs of each attention head are concatenated and then subjected to a further linear transformation to produce the final output of the *Multi-Headed Self-Attention* sublayer.
After the attention sublayer, a *Feed-Forward Neural Network* (FFNN) layer, a fully connected two-layer structure designed to process sequence embeddings independently, is inserted. This sublayer consists of two linear transformations, interspersed with activations of the rectified linear units (ReLUs) between them.
Both sublayers described are followed by a Normalization Layer, whose task is to normalize the sum of the inputs from the sublayer with the outputs generated by the same sublayer. This sublayer is crucial in stabilizing the learning process, helping to avoid the problem of the gradient disappearing through the direct passage of information through the layers.

As highlighted in Figure 3.5, the main difference between the Encoder and the **Decoder** lies in the fact that the latter has two attention sublayers.

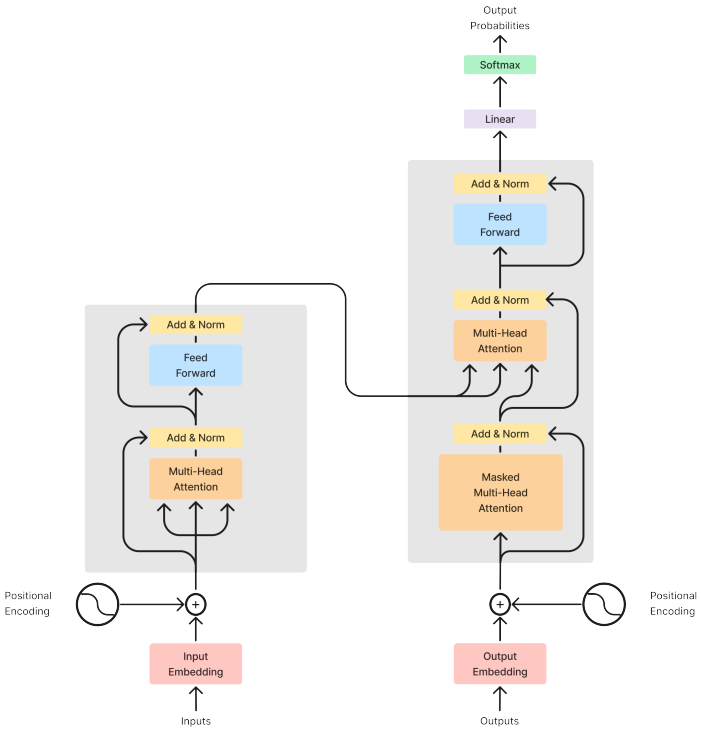

Figure 3.5: The Transformer - model architecture.

In the *encoder-decoder* attention layer, Multi-Head attention is applied to the key and value vectors from the encoder output and the query vector from the previous layers of the decoder. This layer allows the decoder to focus on different parts of the encoder input while generating the sequential output, as was the case in the Seq2Seq models.

Masked Multi-Head Attention, on the other hand, has the same operation as the encoder attention layer but a mask is applied to the attention scores to set the weights corresponding to the next tokens in the sequence to zero, ensuring that only the previous tokens are considered during the attention calculation.

At the end of the decoder sublayers, a linear function is applied to project the output of each token into the dimension space of the vocabulary, followed by a softmax function to obtain a probability distribution over the entire vocabulary. This probability distribution represents the probability of each word in the vocabulary to be the next word in the output sequence [35].

3.2 Question Answering System

A question-answering system is a software application designed to analyze a question posed in natural language, understand its meaning, search for relevant information from a large source of data, and generate an understandable and accurate answer for the user.

These kinds of systems are becoming increasingly important in a wide range of areas. Starting from accessing information online, as evidenced in search engines, to the field of medical care, where they are used to provide immediate and accurate answers in the medical field.
In the course of this section, we will explore in detail the architecture, operation, techniques, different types, and challenges of question-answering systems.

3.2.1 Generic Architecture of a QA model

The architecture of a QA model can be divided into three main components, as shown in Figure 3.6:

- *Question Processing*, which consists of analyzing the question to identify the focus, i.e., the relevant keyword or sequence of words, and classifying the question type to better understand the context. Finally, it rephrases the

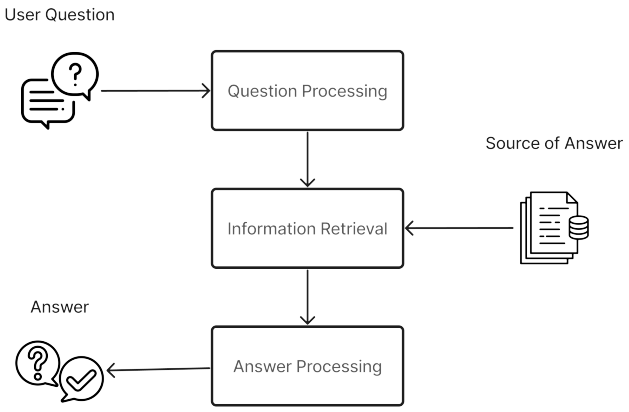

Figure 3.6: General Architecture of a Question Answering System

question, if necessary, for the next information retrieval step [1], using for example Query Logic Language (QLL).

- *Information Retrieval* whose task is to identify relevant information to compose the final answer within a set of documents that can be of different types. This can be done by exploiting different Information Retrieval models, such as the vector model that represents the query and documents as vectors and uses cosine similarity to determine the similarity between the two or more complex models based on deep neural networks such as transformers, which can learn semantic representations of documents and queries [5].

- *Answer Processing* is responsible for the final generation of the response to the user, from the documents returned by the IR model, the most relevant answer is selected, a choice that can be made in different ways depending on the specific architecture of the QA model and the characteristics of the task. Finally, natural language generation techniques produce an understandable and consistent response.

3.2.2 Generative Question Answering System

While traditional question-answer models can extract direct answers from a set of documents or data, generative question-answer models can create entirely new answers based on an understanding of the prompt provided as input. The term "prompt" refers to an instruction given to the model in natural language.
To perform this task, the model must use an LLM, such as a generative neural network, capable of producing sequences of words or tokens that make up the

answer. The main transformer-based models used for this objective and their evolution will be explained below.

3.3 Evolution of Self-Supervised Learning in NLP

In this chapter, the evolution of Self-Supervised Learning in natural language processing is examined. This progression, illustrated in Figure 3.7, has been characterized by three main phases: embedding models, Pretrained Language Model (PLM), and Large Language Model (LLM). The first phase has already been previously introduced and analyzed in Chapter 3.1.1. In the following sections, we will explore the implementation and impact of PLMs and LLMs in the field of natural language.

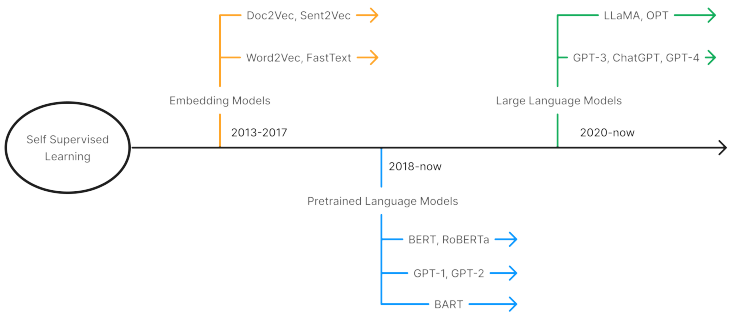

Figure 3.7: Evolution of Self-Supervised Learning in NLP

3.3.1 Pretreined Language Models

The evolution from transformer models to pretrained models was motivated by a significant limitation of the former, which required a large amount of labeled data for training, making the process very expensive and inefficient. However, with the advent of pretrained models, two key concepts were introduced that revolutionized the field: transfer learning and self-supervised learning.

Self-supervised learning is a learning paradigm that aims to exploit inherent structures or relationships within input data to learn discriminative representations for downstream tasks [7]. The use of this technique has proven to be extremely effective in pre-training natural language models, enabling them to achieve state-of-the-art performance on a variety of tasks without the need for explicit supervision,

making the models more adaptable and versatile.

The term **transfer learning** refers to the ability to transfer knowledge from one task or domain to a different but related one [41]. The inherent mechanism is the same as it is for humans who can more easily learn a programming language such as Python from knowledge of another language such as C because the two languages have concepts in common. The use of this learning paradigm makes it possible to reduce the amount of annotated data needed to train a model, improve performance on one task by using pre-learned knowledge on other tasks, and prevent a model from needing training from scratch.

Early Language Models (PLMs) include GPT-1 and BERT, both of which follow the "pretrain then fine-tune" paradigm. According to this approach, the model is first pretrained on an unsupervised learning task, using a large corpus of data, and then adapted to a specific task using a labeled dataset relevant to that task. Although the two models follow the same paradigm they show relevant differences.

BERT is the first encoder-only model based on a transformer architecture; its pretraining is based on two goals: predicting masked tokens in texts and determining whether two text passages follow each other. Unlike previous models, it analyzes the context both before and after a given word during the learning process.

GPT-1 uses a transformer decoder architecture as a feature extractor and is trained on the BookCorpus with 117 parameters. Although the model showed significant improvements over past results, it has some significant limitations: it tends to generate repetitive text, especially if the inputs are different from the data used during training. It has difficulty capturing long-term dependencies in text. The results obtained demonstrated the power of the Generative Pre-trained Transformer models.

GPT-2, follows in the footsteps of the previous model but includes some improvements: larger data collection for pretraining and a larger parameter scale (1.5 million). This led the model to develop a remarkable ability to generate extremely fluent and realistic text compared to its previous counterpart. The increase in model size reduced perplexity without saturation, suggesting that GPT-2 could benefit from longer training. Larger language models could improve natural language understanding and generation, laying the foundation for future models [14].

3.3.2 Large Language Models

PLMs are considered specialized AI systems for specific tasks, but research aims to create versatile models that can generalize to a wide range of tasks. Researchers focus on enhancing computation, data, and model size, leading to the creation of LLMs, an advanced form of PLMs. In this section will examine the LLMs that have demonstrated the best performance in the main natural language processing tasks. These models can be divided into two categories: closed-source LLMs and open-source LLMs.

The evolution of LLMs starts from the **GPT-3** model, which is about 100 times larger than previous models and is trained on a very large text corpus collected from various sources. What makes it unique from its predecessors is the adoption of a new learning paradigm called *"In-Context Learning"* (ICL). In this approach, the model learns new skills from a small set of examples provided directly in the prompt at the time of inference. This instant learning capability, however, does not involve any updating of the model's weights after the response, which means that the acquired knowledge is immediately forgotten [8].
GPT-3, although advanced, has some important limitations: it is not trained on code data, so it has difficulty solving mathematical problems, sometimes misinterprets complex instructions, generates malicious text, and has problems understanding specific contexts.
To overcome these limitations, later models such as GPT-3.5 were developed through the use of supervised fine-tuning (SFT) or reinforcement learning by human feedback (RLHF) techniques applied to the GPT-3 model.
Reinforcement Learning from Human Feedback is a model training technique that aims to align language models with users' intentions. Essentially, it involves using feedback provided by users on model outputs to update model weights through a learning process [23].

Although the latest models have led to the generation of both natural language and code, they are still not optimized for chat conversations. To address this challenge, later models such as **ChatGPT** (also known as GPT-3.5 turbo), which made the GPT-3 model accessible to users, were developed.

GPT-4 represents the latest model released by OpenAI and is capable of performing increasingly complex tasks. Equipped with multimodal capabilities, it can receive as input not only text but also images; in addition, it has a larger context window that allows the model to store more data during a chat session.

Although the exact number of parameters used to train the model has not been released, it is estimated to be in the trillions.

Among the most successful open-source projects is the **Llama 2** series, a collection of models developed by Meta. These models, pre-trained on a wide range of public domain online data sources, represent a benchmark in the field of artificial intelligence.
Their architecture is deeply inspired by that of GPT models, relying on the transformer framework and exploiting sophisticated attention mechanisms. However, Llama models are distinguished by their unique training and optimization methodologies; the models are trained with anywhere from 7 billion to 70 billion parameters.

Chapter 4

AI-Driven XML Generation in ETL

This chapter will examine the project conducted at Mediamente Consulting Srl in more detail. As anticipated, SSIS is one of the company's predominant tools for creating pipelines. However, as there is currently no extension for the automatic generation of ETL pipelines or individual components, work has focused on this gap.
The primary purpose is to speed up the work of corporate employees involved in the ETL flows, providing them with interface access to the model that can generate the SSIS components needed for pipeline design. The activity was divided into:

- Development of the functions associated with the individual components that constitute the ETL flow.
- Drafting the Excel file including the properties and structure of the tables from which the data are taken in the flow.
- Creation of a database containing the company's main user queries on the creation of individual components or flow segments.
- Construction of the pipeline by exploiting embedding techniques, the vector database, and a generative model to produce an executable XML file in SSIS that satisfies the user request.

4.1 SSIS Component Creation

To better understand how the project has been structured, it is first necessary to deepen what are the main data tools used in SSIS.

4.1.1 The primary elements of SSIS

The **Package** represents the fundamental working unit in SSIS and is a logical container of connections, control flow elements, data flow elements, event handlers, variables, parameters, and configurations, necessary to execute the ETL. Within the package there are three main elements:

- **Control flow** defines the order and logic with which operations are carried out in the package
- **Data flow** within which the data extraction, processing, and loading operations are effectively defined
- **Connection managers** manages the connection between the package and the database or other data sources/destinations

The package generally includes a single control flow, housing several types of Tasks and **Containers**. Containers may include one or more Tasks and fall into three categories: For Loop Container, Foreach Loop Container, and Sequence Container. The example shown in Figure 4.1 illustrates the package related to the Staging Area of the ETL flow, which is made up of a Sequence Container that executes two Data Flow Tasks in sequence.

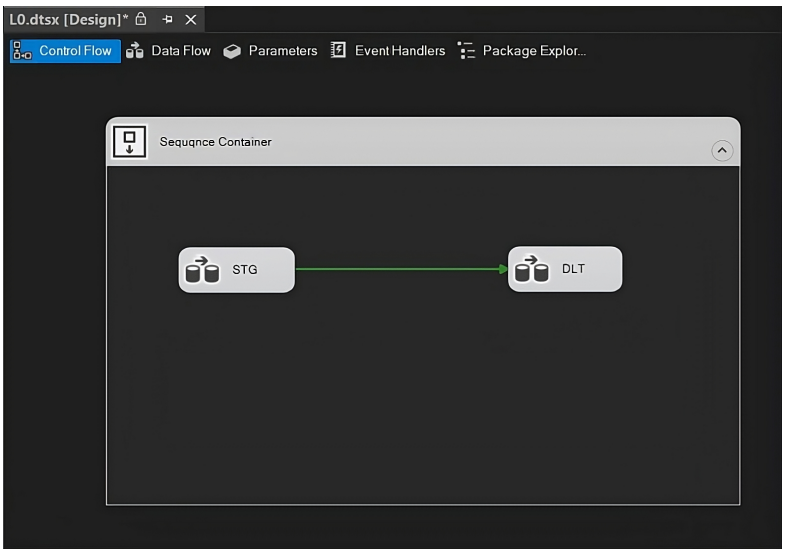

Figure 4.1: Visualization of an SSIS package designed for the Staging Area

Control Flow then acts as an orchestrator, coordinating the execution of Tasks and Containers through the use of precedence constraints [17].

Within the Control Flow, among the numerous available ETL tasks, one that holds particular relevance for the business workflow is the **Data Flow**. This task serves as the core for processing and moving data within an SSIS package, acting as a pipeline mechanism that transfers data from a source to a destination while allowing for data manipulation within the process [17].

Let's now delineate the most commonly used Tasks within Data Flow, represented by source, destination, and transformation components.

The **Source** component of the SSIS package specifies the source location of the data through the use of Connection Managers acting as intermediaries between the package and external data stores. There are various types of Sources, each associated with different connection types, including OLE DB, ADO.NET, Flat File, and many more. The Sources allow both to import the table entirely and portions of the data through the use of a specific SQL command.

The **Destination** component, although it shares many features with the Origin component, differs in its main purpose, which is to load data into a specific Destination. This Destination can be of different types and thus be associated with various forms of connection, similar to the Origin. The Destination also offers the flexibility of loading the entire set of processed data or only a part of it, through the use of SQL commands.

The Tasks used for data transformation are multiple and cover a wide range of functionalities, including operations of split, divert, and remerge, as well as processes of validation, cleaning, and rejection of non-compliant data. In addition, other components dedicated to data transformation are available, such as adding columns, aggregation, and even the use of scripts for applying specific business logic.

An illustration of the application of these three components is showcased in Figure 4.2. In detail, the figure highlights the components employed during the Staging phase development of the flow. The data is extracted from the SRC table, situated in a SQL Server database, utilizing an **OLE DB Source** component adept at retrieving data from an OLE DB-compliant relational database. After data extraction, a modification step follows, in which the `JOBID_LO` field is introduced into the source records, obtained through a **Derived Column** component. The field just introduced in this context is derived from a local variable present within the package; however, in other situations, it is possible to calculate the new field

by using a series of functions or transformations on existing columns. Finally, the newly modified data is loaded into the STG table on SQL Server using the **OLE DB Destination** component.

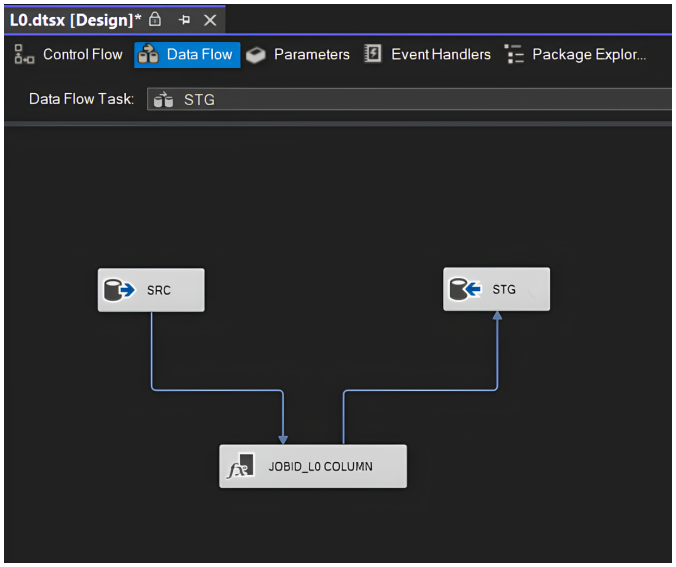

Figure 4.2: Visualization of the Data Flow related to the Staging phase in SSIS

Another example representing the OK phase is depicted in Figure 4.3, in this case, data is loaded from the DLT table and split using the **Conditional Split** component.

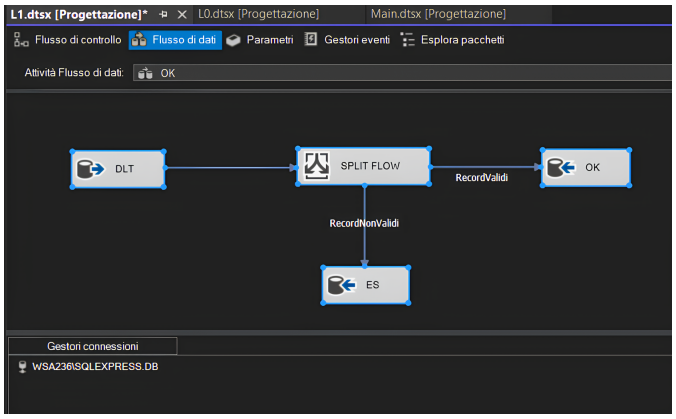

Figure 4.3: Visualization of the Data Flow related to the OK phase in SSIS

This component allows directing data from a single source to different outputs based on conditions defined in the SSIS language [17]. Specifically, in this case, the

data is split based on null fields in the primary keys and directed to a table ES that stores the discarded records. This helps track these rows and potentially modify their content to make them compliant with the rules. If the fields are correct, they are written to the OK table using an OLE DB Destination component.

4.1.2 Design of functions for SSIS component creation

In the context of designing and implementing data integration workflows with SQL Server Integration Services (SSIS), a central element is the files with the ".dtsx" extension. These files, essentially in XML format, provide a structured and detailed representation of data transformation and loading activities and operations within an SSIS project.
The code 4.1 displays the structure of the package component. Within the `<Executables></Executables>` tag, individual SSIS components that need to be executed during the execution of the main SSIS package are specified and configured.

```
<DTS:Executable>
    <DTS:ObjectData>
        <!-- component properties -->
    </DTS:ObjectData>
    <DTS:ConnectionManagers>
        <!-- connection managers used by component-->
    </DTS:ConnectionManagers>
    <DTS:Executables>
        <!-- Other components to be executed -->
    </DTS:Executables>
</DTS:Executable>
```

Listing 4.1: Structure of the package fundamental unit in the DTSX file in XML

Since there are no predefined libraries in Python to automatically generate components in XML format, dedicated functions have been developed for each type of SSIS component. These functions accept the component specification as input and, using this information, generate the corresponding XML code.

The code 4.2 shows the generic structure of a component in SSIS. For `<input></input>` and `<output></output>` tags, specific functions have been defined with varying characteristics depending on the type of component. These functions receive the necessary parameters from an Excel file that describes the structure of

the ETL flow. This file contains detailed information about the tables in the various stages of the flow, including the records in each table and their properties such as data type, length, precision, scale, and primary key indication.
Through the implementation of these functions, Python scripts dedicated to defining SSIS components have been developed. These scripts, which form the core of process automation, will be used to answer various user queries, forming the corpus of our vector database.

```
<component
    refId="Component_ID"
    componentClassID="Component_Class_ID"
    contactInfo="Component_Contact_Info"
    description="Component_Description"
    name="Component_Name"
    usesDispositions="true/false">

    <!-- Inputs of the component -->
    <inputs>
        <!-- Definition of inputs -->
    </inputs>

    <!-- Outputs of the component -->
    <outputs>
        <!-- Definition of outputs -->
    </outputs>
</component>
```

Listing 4.2: Generic Component Structure

Another crucial feature of component representation in SSIS is the `DTSID`, which is a unique identifier assigned to each component, variable, or activity within the package.
This identifier is used internally by SSIS to uniquely refer to the object during package execution. Since the components were created outside of SSIS and therefore the tool cannot automatically generate the identifiers, a function was adopted that, at the beginning of each new user request, generates the identifiers for each item that requests it.bbbb

4.2 Excel Structure

Since the model lacks information about the characteristics of the tables associated with the different stages of the flow, it was deemed essential to create an Excel file describing the main characteristics of the records in each table, following a specific schema. This decision was also influenced by the company's pre-existing automation on the ODI platform regarding mapping development. In this context, automation involves the creation of an Excel file that defines the sequential structure of the mappings, which is subsequently processed by a dedicated algorithm.

The structure of Excel for the pipeline bears similarities to that used in ODI, but some modifications have been made to better suit the specific needs of SSIS. In particular, fields were added to identify the primary keys of tables and to handle expressions related to the computation of new fields, which are propagated from one table to another.
For example, fields such as INS_TIME or $JOBID$ are accompanied by the respective expressions for calculating their values, such as $GETDATE()$ and @[$\$Package :: JOBID$], indicating the retrieval of the value of a variable contained in the package. In addition, the data characteristics of each table used in the flow are specified, including type, length, scale, and precision. This information is crucial for the functions created, as it is essential to specify these details in the XML code.
The creation of Excel represents one of the few manual tasks required of users. However, since its structure is standard, its preparation requires minimal effort.

4.3 Database Construction

To train the model effectively, a specific database was developed in the domain of interest, focusing on requests made by company employees to create individual components or entire flow segments in SSIS.
The compilation of potential queries stemmed from an analysis of employee requirements, concentrating on prevalent requests encountered during workflow development. For instance, a typical user query might involve *'Construct a package component with a data flow comprising an OLEDB source task'* alongside more intricate demands like *'Create the complete flow structure of Level 0'*.

Despite efforts to cover a wide range of requirements, it is important to note that the database does not include every possible component that can be created in

SSIS, only the most common and required components for flow development have been included.

All questions were collected in the file "question.txt," while a second database called "answer.txt" contains the Python scripts associated with each question. These scripts containing the functions needed for the specific request generate the desired response as an XML file.

4.4 Pipeline Implementation

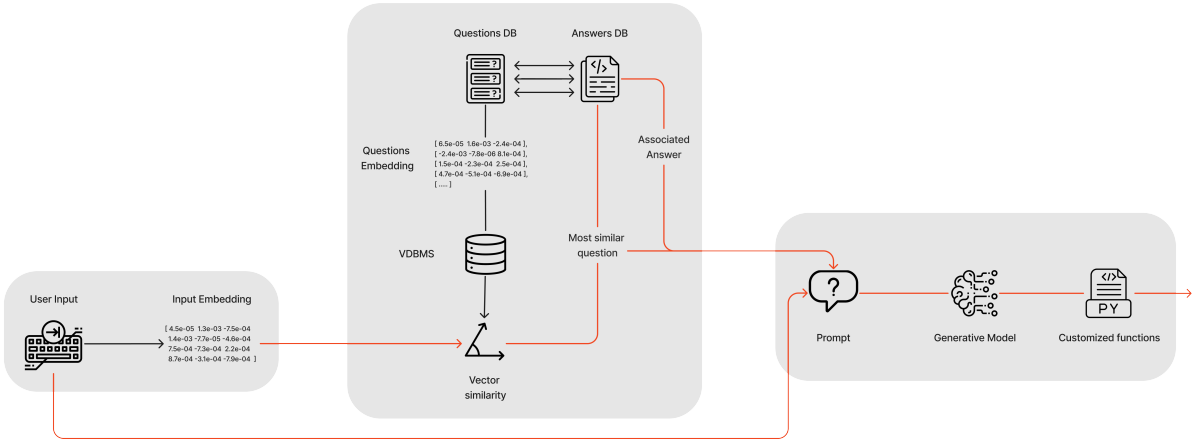

Figure 4.4: Structural overview of the pipeline for XML file generation

After outlining the fundamental elements needed to build the components and meet user requirements, a detailed description of the pipeline construction is given, as illustrated in Figure 4.4. In the initial stage of the pipeline, the Excel file is imported and the table names, used through the database connection, are extracted. This step is crucial for extracting and loading data during the various stages of the flow and within the different components.
Next, questions and their answers are extracted from the corresponding *"question.txt"* and *"answer.txt"* files. This data is then loaded inside a dictionary, where each question is associated with its corresponding answer.

The Python code associated with the created pipeline and all tests conducted on the different models were implemented and executed on Google's Colab platform. This choice was driven by the availability of advanced computing resources that

Colab provides, its ability to avoid the need to configure local development environments, and the completely free nature of the service.

The next subsections will outline the main steps for implementing the pipeline. At the same time, Chapter 4.5, on experiments and results, will detail the technologies used and the motivations behind the choice of specific models. In the Algorithm 1, the main steps of the pipeline are described in detail.

4.4.1 Optimizing Text Embedding and Query Retrieval

The Word2Vec model, recognized as one of the best for text embedding, transforms user input into a vector through an embedding process. In addition, this same process has been applied to the query database, resulting in a vector database management system (VDBMS). Then, through the application of a similarity metric, in this case, cosine, the query most similar to the user's query among those in the VDBMS is determined.
To optimize the performance of the embedder, the technique of **semantic weighting** was implemented; this technique involves assigning a higher weight within the embedding vector to the words identified as most significant.

Through this method, the answer most akin to the user's input query is identified. Consequently, the Python script containing the functions to generate the XML file most closely matches the user's request.

4.4.2 Prompt Construction

After identifying the question most similar to the one posed by the user and its answer, a fundamental step for successful automation is reached: the creation of the prompt.
This stage is crucial because crafting a well-structured prompt enables the GPT model, trained on various tasks, to discern the user's needs and generate the desired outputs. Therefore, understanding the various prompt techniques to optimize interactions with the generative model becomes essential:

- **Zero-Shot Prompting :** The model is instructed through a prompt describing the task to be performed, but without having access to labeled data for training on precise input-output mappings. Therefore, the model relies on its pre-acquired knowledge to come up with predictions based on the prompt provided for the specific task.

Algorithm 1 XML File Generation Algorithm

Require:
input: User Request
questions: Questions Database
answers: Answers Database
weight_dictionary: Dictionary of word weights

Ensure:
XML file

```
function WEIGHTED_EMBEDDING(model, question, weight_dict)
    for word in question.split() do
        if word in model.wv then
            weighted_vector ← model.wv[word] * weight_dict.get(word, 1.0)
            weighted_vectors.append(weighted_vector)
        end if
    end for
    return np.mean(weighted_vectors, axis=0).reshape(1, -1)
end function

word2vec_model ← Word2Vec(questions, vector_size=50, window=5,
    min_count=1, sg=0)
input_vector ← WEIGHTED_EMBEDDING(word2vec_model, input,
weight_dictionary)
VDBMS ← [ WEIGHTED_EMBEDDING(word2vec_model, question,
weight_dictionary) .flatten() for question in questions]
id ← argmax(cosine_similarity(input_vector, VDBMS))

gpt_prompt ← f "
    To tailor the response to the new question {input},
    lets begin by analyzing the similar question {questions[id]},
    whose answer is {answers[id]}.
    After identifying the parameters to be changed, modify the response to
    adapt it accordingly. Generate three potential responses to the new
    question following the reasoning made above.
    Select the response that best fits the users specific query "

message ← { "role": "user", "content": gpt_prompt}
output ← client.chat.completions.create( model="gpt-4-0125-preview",
    textmessages = message, temperature=0.5,
    max_tokens=500, frequency_penalty=0.0)

exc(output.choices[0].message.content)
root ← ET.fromstring(result)
tree ← ET.ElementTree(root)
XML_file ← tree.write("result.xml")

return XML_file
```

In this type of technique, for particularly detailed or complex tasks, the accuracy of responses may not be optimal [26].

Zero-Shot Prompting

Prompt: "Classify the text into positive, neutral, or negative:
Text: That shot selection was awesome.
Classification:"

Example of One-Shot Prompt

- **Few-Shot Prompting :** This technique provides models with a limited number of input and output examples to help them better understand a given task. The use of few examples, as long as they are of high quality, is effective in improving model performance in complex tasks.
  However, this strategy requires the use of more tokens to include these examples, which can be particularly resource-intensive when working with longer texts [3].

Few-Shot Prompting

Prompt: "A 'whatpu' is a small, furry animal native to Tanzania. An example of a sentence that uses the word whatpu is: We were traveling in Africa and we saw these very cute whatpus.
To do a 'farduddle' means to jump up and down really fast. An example of a sentence that uses the word farduddle is:"

Example of Few-Shot Prompt

- **Chain-of-Thought (CoT) Prompting :** This technique leads the model through a sequence of logical steps, deconstructing the main problem into a series of intermediate steps, and facilitating the achievement of a final answer.
  Compared with traditional prompts, this allows the model to enhance its performance in a variety of tasks, including arithmetic, common sense, and symbolic reasoning tasks [39].

CoT Prompting

Prompt: " Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.
Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?
A: "

Example of CoT Prompt

This kind of prompt requires manual dexterity in devising effective and varied examples, thus involving significant effort.
To mitigate this need, Automatic Chain-of-Thought (Auto-CoT) was introduced, a two-step process: initially, questions are grouped into similar clusters; then, a representative question is chosen from each cluster and a sequence of reasoning is generated using Zero-Shot-CoT with simple heuristics [40].

- **Self-Consistency :** This prompting strategy improves reasoning skills over simple greedy decoding within the Chain of Thought. It involves repeatedly executing the same language model on the same prompt, ultimately selecting the most consistent result as the final response, thus adopting a "self-ensemble" approach.
  This methodology allows different reasoning paths within the Chain of Thought to be explored, allowing the most accurate one to be selected, this enhances the performance of CoT prompting in reasoning tasks.

- **Rephrase and Respond (RaR) Prompt :** This approach has been devised to bridge the gap between human thought structures and those of LLMs. The method involves reformulating the original question, incorporating additional details to enhance semantic clarity and address inherent ambiguities within the question, thereby enabling the model to respond more accurately [6].
  An example is illustrated below, where the prompt: "Was {person} born in an even day?" has been rephrased using the RaR technique.

RaR Prompting

Prompt: "Could you provide more information on whether the individual named {person} was born on a day that is an even number? This refers to dates such as the 2nd, 4th, 6th, 8th, and so on within a given month."

Example of RaR Prompt

The prompt engineering techniques mentioned represent only a limited subset of the many methods available. In particular, the selected techniques are those considered most congenial to the ultimate goal of this project.
The various techniques differ in the application domains with which they are associated, the type of applications they are intended to support, the method of prompt acquisition, and the number of prompts used during interaction with the language model. These differences are made explicit in detail in the Table 4.1

Prompting Technique	Application	Prompt Acquisition	Prompt Turn
Zero-shot	new tasks without training data	manual	single
Few-shot	new tasks without training data	manual	single
CoT	reasoning and logic	manual	multi
Self-Consistency	reasoning and logic	manual	single
RaR	understanding user intent	manual	single

Table 4.1: Summary of LLM prompting techniques by factors: application, prompt acquisition, prompt turn

4.5 Experiments and Results

This chapter will detail the different techniques and models used and tested within the pipeline, and compare the selected parameters and their results obtained to select the optimal solution for the pipeline.

4.5.1 Evaluating Text Embedding Models

As previously illustrated, selecting the optimal text embedding model is crucial to ensure the identification of the functions best suited to effectively respond to the user's request. This step is critical to provide the generative model with relevant

examples, thus enabling it to tailor its functionality specifically to the user's query and correctly learn the task to be performed.
A detailed analysis of the main models, including those tested for the pipeline, was provided in the Chapter 3.1.1.

Systems employing embedding for text matching have a significant limitation: while capable of capturing semantic features of text, they can be affected by text length, word distribution, and other factors that do not necessarily reflect semantic intention or context.
To mitigate this effect, there are several approaches available; in the context of this application, **semantic weighting** was adopted. This approach involves introducing a higher weight within the embedding vector for specific words of higher relevance. This ensures that terms of greater importance have a higher impact in establishing the similarity between the user's query and the collection of database queries.

Three key metrics were considered to evaluate different embedding models: accuracy, recall@K and precision@K. These metrics provide a comprehensive overview of model performance in information retrieval or classification scenarios.

The **accuracy** measures the percentage of times the model selects exactly the correct question among all possible options.

$$Accuracy = \frac{\text{Number of correct question selections}}{\text{Total number of questions}}$$

Where a "correct question selection" means that the model has identified the question that exactly matches the Python function for generating the user's expected component.

Given the possibility that multiple queries may seem similar to the user's query but only some of them are truly relevant, **precision@K** evaluates how much of the first K queries selected by the model are relevant. A high value means that, among the K questions that the model considers most similar, most are relevant to the user's query.

$$Precision@K = \frac{\text{Number of relevant questions among the first K selected}}{\text{K}}$$

Recall@K is important to assess whether the model can "retrieve" or identify

the correct question among its top K choices, considering all relevant questions in the dataset. In practice, if there are multiple versions of a question that could be considered correct, recall@K tells us whether the model can find them within its top K choices.

$$Recall@K = \frac{\text{Number of relevant questions found in the first K selections}}{\text{Total number of relevant questions in the dataset}}$$

For the parameter K, the value 3 was selected. This choice aims to increase the accuracy of the choice, ensuring relevant selection of the most similar questions without overloading the user. Considering that each component in the database has 3-5 related queries associated with it, $K = 3$ optimizes the precision of the answers provided, which is crucial for correctly identifying the Python functions of the component.
Table 4.2 presents results of different embedding models, with and without semantic weighting, 25 possible user queries have been tested.

Model	**SW**	**Accuracy**	**Precision@3**	**Recall@3**
Word2Vec	No	0.76	0.61	0.52
	Yes	**0.96**	**0.72**	**0.6**
FastText	No	0.48	0.42	0.39
	Yes	0.64	0.58	0.51
TF-IDF	No	0.68	0.53	0.46
	Yes	0.88	0.67	0.56

Table 4.2: Performance comparison of embedding models with and without semantic weighting on Accuracy, Precision@3, and Recall@3 metrics

As demonstrated by the results obtained, the use of embedding models in combination with semantic weighting shows a performance improvement.
This improvement can be explained by considering the example of a typical user request, *"generate a flow with name FLOW with inside an origin named SOURCE"*. Using the Word2Vec model without semantic weighting, the most related response identified is: *"generate a data flow component named STG with inside a derived column named DC"*. In contrast, the Word2Vec model enriched with semantic weighting identifies the most relevant response: *"create an origin named STG inside a flow named DLT inside a sequence named SEQ"*, which better reflects the

user's initial request.
Through this approach, the model places less importance on word order or array length, focusing instead on key terms such as *"origin"*, which essentially captures the user's desired component.

Several measures of similarity and distance were evaluated, including cosine similarity, Euclidean distance, and Manhattan distance. The corresponding formulas are shown in Table 4.3, where **a** and **b** denote the vectors under comparison.

Formula	**Definition**
Cosine Similarity	$\text{sim}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|}$
Euclidean Distance	$d(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$
Manhattan Distance	$d(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^{n} \|a_i - b_i\|$

Table 4.3: Formulas for Cosine Similarity, Euclidean Distance, and Manhattan Distance

Among the three options considered, cosine similarity emerges as the preferred choice; this metric evaluates the angle formed by two vectors in space, making it independent of their length.
In contrast, Euclidean distance assumes significance when operating in a space whose dimensions are of equal importance, its use can lead to the problem of the *"curse of dimensionality"*, where sparsity of data and high dimensionality cause distortions in the results.
Manhattan distance proves to be more suitable than Euclidean distance in environments characterized by high dimensionality or sparse data, and is particularly effective in scenarios where discrepancies in each dimension linearly affect the outcome, nevertheless, it remains sensitive to the size of the vectors [38].

4.5.2 Analysis and Selection of the Generative Model

The core element of the pipeline is the generative model, employed to adapt functions to user requests. Several models, both open-source and proprietary, have been tested to identify the best one.

In the open-source context, one of the models in the Llama 2 series, a suite of pre-trained and optimized generative language models developed by Meta, was

analyzed. Specifically, the llama-2-13b-chat.ggmlv3.q5_1.bin model was chosen, one of the latest released in this collection, trained with 13 billion parameters and with a size of 9.76 GB.

On the proprietary model side, three models developed by OpenAI were examined: gpt-4-0125-preview, gpt-4-0613, and gpt-3.5-turbo-0125. Access to all three models was through the API provided by OpenAI, thus eliminating the need to download models locally, which would have consumed considerable storage space and resulted in additional complexity in managing them.

The models were tested based on 25 questions, using the same prompt, the results achieved, along with several characteristics of the models such as API costs, context window size, and accuracy of responses, were reported in Table 4.4.

Model	Type	Accuracy	Input price	Output price	Context window
llama-2-13b-chat.ggmlv3.q5_1.bin	Open-source	0.65	$0	$0	4096
gpt-3.5-turbo-0125	Closed-source	0.76	$0.0005 / 1K tokens	$0.0015 / 1K tokens	16385
gpt-4-0613	Closed-source	0.79	$0.03 / 1K tokens	$0.06 / 1K tokens	8192
gpt-4-0125-preview	Closed-source	**0.85**	$0.01 / 1K tokens	$0.03 / 1K tokens	128000

Table 4.4: Comparison of text generation models: type, accuracy, API cost for input prompt and response every 1K tokens, and context window size.

From the table it seems clear that the model with the best results is gpt-4-0125, also known as **GPT-4 Turbo**, in addition to having higher accuracy in responses, which is essential for the proper execution of functions that require to be generated accurately to be executed and build a correct XML file, it also has significantly lower costs than the previous model.
The only open-source model tested showed difficulties in understanding the required task, specifically in understanding the structure of the functions and their parameters and thus failing to correctly substitute parameters within them.

It was decided to examine several GPT models, as they are currently the cutting edge of the state of the art in the field of text generative models. Similarly, the Llama model was identified as a benchmark among open-source models.

4.5.3 Prompt Engineering vs Fine-Tuning

In this subchapter, we explore the challenge of adapting the model, which was not originally trained to recognize and generate the specific features designed for

component creations.

When faced with this situation, it becomes crucial to determine the optimal approach: on the one hand, there is the option of **fine-tuning**, which involves additional training of the pre-existing model to refine its capabilities to our specific needs; on the other hand, there is the option of **prompt engineering**, which is the art of formulating accurate and detailed prompts that guide the model to understand and perform the required task without the need for additional training. Choosing between these two methodologies is critical to the project's success and requires careful evaluation of their implications and potential.

The comparison between the two techniques focuses on several key aspects such as flexibility, cost, customization, and data requirements.
Fine-tuning allows a preexisting model to be customized to deal with highly specialized tasks, thereby improving its capabilities in those specific areas. Through this process, the model can assimilate the peculiarities and various gradations present in the dataset at its disposal.
However, from an economic point of view, fine-tuning involves a significant investment in training resources, especially when working with large datasets. The size of the dataset is crucial not only to prevent the risk of overfitting but also to ensure an effective training process.

The prompt engineering technique allows the model to be oriented toward specific tasks without having to change its architecture or weights, thus offering a faster method because it eliminates the need for an additional learning process. This approach is particularly useful when available training data is scarce, as it requires a limited amount of data to teach the model new tasks.
However, the limitations of this technique lie in its heavy reliance on the ability to formulate effective prompts: if the prompts are not well designed, performance may not be optimal. Also, if the underlying model lacks knowledge in a specific domain, prompt engineering may prove less effective in guiding learning to the desired area.

Both techniques were tested, and the code used to train the model by fine-tuning is described in Algorithm 2.

The analysis of the algorithm shows that the dataset consisting of questions and answers was processed to generate various sequences of demonstration conversations, which reflect the type of interactions that the model is expected to handle during inference. The model chosen for this task is *gpt-3.5-turbo-0613*, reported as

Algorithm 2 Fine-tuning Algorithm

Require:
questions: Questions Database
answers: Answers Database
user_message: User Question in message format

Ensure:
Fine-tuned model

```
function DATASET_LINE(question, answer):
    message ← { "messages":[
     {"role": "system", "content": system_message},
     {"role": "user", "content": question},
     {"role": "assistant", "content": answer} ] }
    return message
end function

dataset ← []
for each question in questions and answer in answers do:
    dataset.append(DATASET_LINE(question, answer))
end for

training_file_id ← openai.files.create(
     file = open(dataset, "rb"),
     purpose ="fine-tune")
validation_file_id ← openai.files.create(
     file =open(dataset[-20:], "rb"),
     purpose ="fine-tune")

response ← openai.fine_tuning.jobs.create(
     training_file = training_file_id,
     validation_file = validation_file_id,
     model ="gpt-3.5-turbo-0613",
     suffix ="q&a-test" )

job_id ← response.id
fine_tuned_model_id ← openai.fine_tuning.jobs.retrieve(job_id)
.fine_tuned_model
user_answer ← openai.chat.completions.create(
     model = fine_tuned_model_id,
     messages = user_message,
     temperature = 0,
     max_tokens = 100 )

return fine_tuned_model_id
```

the most effective for fine-tuning, according to the recommendations in the OpenAI documentation. This choice is motivated by the demonstrated superiority of the model in terms of performance and ease of use.

Although the results obtained from the model trained with fine-tuning are consistent, the accuracy is lower than that obtainable through the direct use of the prompt. This discrepancy is mainly attributable to the limited amount of question-response pairs in the dataset, which does not allow the model to optimally assimilate the correlation of each function with its respective component.

Hyperparameters, such as the number of epochs, learning rate multiplier, and batch size, were kept at the default values recommended by OpenAI.
This decision is based on the observation that the model tends to fit the training data adequately and converges satisfactorily to the target responses, thus avoiding the risk of overfitting, whereas the limitations observed in the fine-tuned model seem to be attributable solely to the scarcity of training data.

Prompting Technique		**Avg # of Tokens**	**Avg Cost ($)**	**Accuracy**	**Avg Time (s)**
Zero-Shot	Normal	205.14	0.004	53%	4.88
	With Rephrase	225.62	0.003	65%	3.5
	With Self-Consistency	452.90	0.011	63%	17.57
Few-Shot	Normal	283.38	0.004	61%	4.18
	With Rephrase	313.57	0.004	75%	3.35
	With Self-Consistency	528.14	0.011	81%	15.35
Chain-of-Thought	Normal	425.62	0.01	57%	14.13
	With Rephrase	385.76	0.008	77%	11.21
	With Self-Consistency	633.90	0.016	**85%**	24.25

Table 4.5: Performance comparison of different prompting techniques tested based on the average number of tokens, average prompt cost, accuracy, and average inference time.

In the prompt engineering approach, several prompting techniques and their various combinations were tested, which are presented in detail in Chapter 4.4.2.
Through a sample of 20 questions, the following were calculated: the average number of tokens used in formulating the prompt and response, along with the average cost associated with the input prompt and generated response, the accuracy of

the different methodologies, and the average inference time for the response, the results are shown in Table 4.5.
These results are obtained from the combination of the generative model and the embedding model selected in the previous chapters, 4.5.2 and 4.5.1, together with the various prompting techniques examined.

Reviewing the results in the table, it is clear that the techniques combined with Rephrase or Self-Consistency show a significant performance improvement. It is important to note that this improvement is accompanied by an increase in cost and time to compute the response; since both prompts created and model results will require more tokens.
However, the costs are still low when compared to the significant time savings in the work of creating the components.

Breaking down the query into subproblems and analyzing the different features that make up the answer allows the model to more thoroughly understand the correlations between the query identified as most similar to the user's input and the input itself. This allows the model to construct the features to be returned correctly and consistently.

For example, considering the user's question: *'create a union tool in a data flow named DATA_FLOW'* and the most similar question identified in the database: *'create a union component called UNION with inside a sequence named SEQUENCE and a package named PACKAGE'* with the corresponding answer:

```
union = union('SEQUENCE', 'Flow_name', 'UNION');
data_flow = createDataFlow(union, 'SEQUENCE', 'Flow_name', '',
generate_id());
sequence = createSequence('SEQUENCE', data_flow, '',
generate_id());
result = createPackage('PACKAGE', sequence, generate_id(),
generate_id(), '');
```

Many prompt techniques have shown a limitation in recognizing the importance of the functions that create the sequence and package components; since this information is not explicitly requested in the user's query.
Prompt methodologies such as Few-Shot and Chain-of-Thought, due to the availability of multiple examples and the ability to reason logically, can capture the importance of these fundamental elements. Moreover, they understand within

functions the meaning of different parameters and, consequently, can determine which ones to modify.

In constructing the pipeline, it was therefore chosen to combine two prompting techniques, Chain-of-thought and Self-Consistency, since they have been shown to produce the highest level of accuracy.

Chapter 5

Conclusion and future implementations

In this thesis work, one of the most significant challenges in data engineering was addressed: the automation of ETL pipeline creation in response to specific user requests. The importance of this research lies in the growing need in business contexts to process and analyze large volumes of data in an efficient, rapid, and personalized manner.
This was achieved through the combination of the capabilities of embedding models, the latest generative models, and various prompt techniques, leading to the results shown in Table 4.5.

The selection of the most suitable generative model and the careful choice of the best embedder represent two crucial steps that significantly affected the accuracy of the responses generated by the system.

The process of identifying the request, among those pre-existing in the database, that most closely matches the user's request and adapting the corresponding function to specifically address the initial query, through the use of a Large Language Model, proved to be a successful approach. This method made it possible to customize functions to the user's needs, which, when executed, return flow components in XML code tailored precisely to the expressed requests.

However, the developed implementation, despite its progress, shows significant limitations. The domain of possible functions, and therefore of generatable ETL components or flow segments, as well as of user-manageable requests, is confined exclusively to those pre-existing elements in the created database. This restriction

limits the ability to adapt to new requirements not previously considered, demanding continuous updating of the database of queries and their responses according to employee needs.

In conclusion through the development of an advanced system, it has been proven that it is possible to significantly simplify the ETL pipeline setup process, reducing development time and human error, as well as allowing for greater flexibility and adaptability to specific user requirements.

5.1 Future Work

This section will explore possible directions the research might take as a result of the findings and conclusions presented previously.

A major aspect of the planned next developments is the **expansion of the vector database**, which is a crucial element in ensuring the effectiveness of the results. Increasing the size of the database dedicated to questions and the associated functions would contribute significantly to improving the accuracy of the model by being able to select questions that increasingly match those posed by the user and ensure a more accurate answer.

A more extensive database would also open up the possibility of **employing fine-tuning** techniques, teaching the LLM the correlation between query and function to execute for obtaining whole chunks of flow. Currently, the model works through a predefined database of questions to identify and select the one most akin to the user's query. This approach, however, narrows the domain of possible questions the model can answer, limiting its flexibility to predefined examples.
Implementing fine-tuning could greatly expand its comprehension capabilities, allowing the model to autonomously interpret and respond to user queries without having to depend on particularly elaborate prompts or examples that trace the formulated question.
This would allow users to formulate increasingly complex questions and be able to generate flow sequences not currently represented in the database, thus enriching the interactivity and effectiveness of the model.

Currently, **automated management of database connections** has not been implemented. This functionality is crucial to allow users to use their connections to access the databases of their interest; users must manually configure these con-

nections through SSIS.

The **automation of the Excel** spreadsheet creation process, which is used to map the structure of tables and sources of data provided to components, would achieve a level of total automation of the project. Currently, this document is still compiled manually by users, even though its standardized structure makes it relatively easy to draft.

Finally, in the future, it would be useful to **develop a more user-friendly interface** to make it easier for employees to access and interact with the system.

Bibliography

[1] Ali Mohamed Nabil Allam and Mohamed Hassan Haggag. "The question answering systems: A survey". In: *International Journal of Research and Reviews in Information Sciences (IJRRIS)* 2.3 (2012).

[2] Piotr Bojanowski et al. *Enriching Word Vectors with Subword Information.* 2017. arXiv: 1607.04606 [cs.CL].

[3] Tom Brown et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[4] Samuel S Conn. "OLTP and OLAP data integration: a review of feasible implementation methods and architectures for real time data analysis". In: *Proceedings. IEEE SoutheastCon, 2005.* IEEE. 2005, pp. 515–520.

[5] Eduardo Gabriel Cortes et al. "A systematic review of question answering systems for non-factoid questions". In: *Journal of Intelligent Information Systems* (2022), pp. 1–28.

[6] Yihe Deng et al. "Rephrase and respond: Let large language models ask better questions for themselves". In: *arXiv preprint arXiv:2311.04205* (2023).

[7] Carl Doersch and Andrew Zisserman. "Multi-task self-supervised visual learning". In: *Proceedings of the IEEE international conference on computer vision.* 2017, pp. 2051–2060.

[8] Qingxiu Dong et al. "A survey for in-context learning". In: *arXiv preprint arXiv:2301.00234* (2022).

[9] Alexandra Maria Ioana Florea, Vlad Diaconita, and Ramona BOLOGA. "Data integration approaches using ETL." In: *Database Systems Journal* 6.3 (2015).

[10] Georgia Garani and Sven Helmer. "Integrating star and snowflake schemas in data warehouses". In: *International Journal of Data Warehousing and Mining (IJDWM)* 8.4 (2012), pp. 22–40.

[11] Carlo Ghezzi. "Designing data marts for data warehouses". In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 10.4 (2001), pp. 452–483.

[12] Zellig S Harris. "Distributional structure". In: *Word* 10.2-3 (1954), pp. 146–162.

[13] William H Inmon. *Building the data warehouse*. John wiley & sons, 2005.

[14] Katikapalli Subramanyam Kalyan. "A survey of GPT-3 family large language models including ChatGPT and GPT-4". In: *Natural Language Processing Journal* (2023), p. 100048.

[15] Kazuya Kawakami. "Supervised sequence labelling with recurrent neural networks". PhD thesis. Technical University of Munich, 2008.

[16] Ralph Kimball and Margy Ross. *The data warehouse toolkit: the definitive guide to dimensional modeling*. John Wiley & Sons, 2013.

[17] Brian Knight et al. *Professional Microsoft SQL Server 2012 Integration Services*. John Wiley & Sons, 2012.

[18] Larry R Medsker and LC Jain. "Recurrent neural networks". In: *Design and Applications* 5.64-67 (2001), p. 2.

[19] Timothee MicKus. "On the Status of Word Embeddings as Implementations of the Distributional Hypothesis". PhD thesis. Université de Lorraine, 2022.

[20] Natalia Miloslavskaya and Alexander Tolstoy. "Big data, fast data and data lake concepts". In: *Procedia Computer Science* 88 (2016), pp. 300–305.

[21] Filippo La Noce and Luigi D'Ercole. *Data Warehousing: Dal Dato all'informazione*. Franco Angeli, 2001.

[22] *Oracle Data Integrator*. (accessed January 04, 2024). URL: https://www.oracle.com/it/middleware/technologies/data-integrator.html.

[23] Long Ouyang et al. "Training language models to follow instructions with human feedback". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 27730–27744.

[24] Matteo Pagliardini, Prakhar Gupta, and Martin Jaggi. "Unsupervised learning of sentence embeddings using compositional n-gram features". In: *arXiv preprint arXiv:1703.02507* (2017).

[25] Steven Piantadosi. "Modern language models refute Chomsky's approach to language". In: *Lingbuzz Preprint, lingbuzz* 7180 (2023).

[26] Alec Radford et al. "Language models are unsupervised multitask learners". In: *OpenAI blog* 1.8 (2019), p. 9.

[27] G Satyanarayana Reddy et al. "Data Warehousing, Data Mining, OLAP and OLTP Technologies are essential elements to support decision-making process in industries". In: *International Journal on Computer Science and Engineering* 2.9 (2010), pp. 2865–2873.

[28] Jasna Rodic and Mirta Baranovic. "Generating data quality rules and integration into ETL process". In: *Proceedings of the ACM twelfth international workshop on Data warehousing and OLAP*. 2009, pp. 65–72.

[29] William Rowen et al. "An analysis of many-to-many relationships between fact and dimension tables in dimensional modeling". In: *International Workshop on Design and Management of Data Warehouses (DMDW 2001), Interlaken Switzerland*. 2001, pp. 1–13.

[30] Shaker H Ali El-Sappagh, Abdeltawab M Ahmed Hendawi, and Ali Hamed El Bastawissy. "A proposed model for data warehouse ETL processes". In: *Journal of King Saud University-Computer and Information Sciences* 23.2 (2011), pp. 91–104.

[31] *SQL Server Integration Services*. (accessed January 04, 2024). URL: https://learn.microsoft.com/it-it/sql/integration-services/sql-server-integration-services?view=sql-server-ver16.

[32] Kunjian Sun and Yuqing Lan. "SETL: A scalable and high performance ETL system". In: *2012 3rd International Conference on System Science, Engineering Design and Manufacturing Informatization*. Vol. 1. IEEE. 2012, pp. 6–9.

[33] Lewis Tunstall, Leandro Von Werra, and Thomas Wolf. *Natural language processing with transformers*. " O'Reilly Media, Inc.", 2022.

[34] Marshall W Van Alstyne, Geoffrey G Parker, and Sangeet Paul Choudary. "Pipelines, platforms, and the new rules of strategy". In: *Harvard business review* 94.4 (2016), pp. 54–62.

[35] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[36] Manole Velicanu, Gheorghe Matei, et al. "Database Vs Data Warehouse". In: *Revista Informatica Economică* 3.2007 (2007), p. 43.

[37] Nithin Vijayendra. "A courseware on ETL process". PhD thesis. California State University, Sacramento, 2010.

[38] Jiapeng Wang and Yihong Dong. "Measurement of text similarity: a survey". In: *Information* 11.9 (2020), p. 421.

[39] Jason Wei et al. "Chain-of-thought prompting elicits reasoning in large language models". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 24824–24837.

[40] Zhuosheng Zhang et al. "Automatic chain of thought prompting in large language models". In: *arXiv preprint arXiv:2210.03493* (2022).

[41] Fuzhen Zhuang et al. "A comprehensive survey on transfer learning". In: *Proceedings of the IEEE* 109.1 (2020), pp. 43–76.

List of Figures

List of Tables

Acknowledgements

I would like to express my sincerest gratitude to Professor Apiletti for his guidance, and support while writing this thesis.
Special thanks go to the team at MediaNe Consulting for offering me the opportunity to conduct my thesis at their company. Your support, helpfulness, and cooperation enriched my experience.
In addition, I would like to extend my heartfelt thanks to all those who have been close to me during this journey. Your presence, encouragement, and support have made the moments of study and work lighter and more fulfilling.