POLITECNICO DI TORINO

"ICT 4 Smart Societies"
course of the Master Degree in
Telecommunications Engineering.

Master Degree Thesis

Automation of ETL Pipelines in DataStage

Academic Tutor:
prof. Albertengo Guido
Company Tutor:
Chiarello Donato

Candidate:
Benedetti Alex Umberto

Academic year 2024-2025

Ai miei genitori,
che hanno permesso la
realizzazione di tutto
questo .

Abstract

In today's business ecosystem for enterprise data management, the automation process of ETL (Extract Transform Load) pipelines has become one of the primary objectives. This master's thesis explores the use of latest Artificial Intelligence techniques to simplify data integration and transformation, with the aim of minimizing the human effort in designing and managing workflows to process data.

Conducted in partnership with Mediamente Consulting Srl, this research aims to design and implement a system that examines and utilizes advanced technologies to effectively manage user requests in an ETL data flow context.

The central process involves the automation of DataStage components using XML templates to dynamically create and configure jobs based on the interpreted user requests. Through the development of custom scripts, the system automates the deployment and configuration of DataStage jobs, transforming the ETL setup from a manual, error-prone process into a more efficient and reliable automated procedure. It employs complex methods of data representation techniques designed to capture the semantic nuances and contextual elements present in the queries. These distributed representations represent the basis in order to finding the most appropriate ETL solution between a set of different available options presented. The selected solution is then analyzed by a generative model, again adapting it to the original specifications and thus enhancing the overall relevance and coherence of the final outcome.

In developing the proposed pipeline, different embedding techniques and generative models are analyzed and tested. The most effective methods are selected based on their ability to provide answers that closely adapt to the user's needs, as discussed in the thesis.

The application of this methodologies yields an optimized ETL configuration which closely adapt to the user needs, minimizing the manual configuration in the ETL tasks. These type of operations not only improve operational efficiency but also allows enterprises to respond more dynamically to changing data requirements.

Acknowledgements

Table of Contents

Chapter 1

Introduction

This thesis is the result of an internship experience at Mediamente Consulting S.R.L., an IT consulting company that specializes in data management and enhancement. Mediamente Consulting is an integral part of VAR Group.
VAR Group, a well-established conglomerate in the IT sector, comprises various companies specialized in different aspects of information technology and digital transformation. The company, with offices in Turin, Bologna, Milan and Rome, is organized into different business units: Data Integration, Data Science, Data Visualization, Corporate Performance Management and Advanced Analytics.
Together, these units provide clients with IT infrastructures that manage, organize and analyze complex data, with the goal of transforming it into useful information for strategic business decisions.

1.1 Objectives and structure of the thesis

Effective data management today represents a crucial element for any organization aiming to maintain competitiveness in the market and to facilitate strategic decision-making. The quality, integrity and accessibility of data play a fundamental role in supporting quick and effective strategic decisions, thus emphasizing the importance of robust data handling practices.

The central focus of this thesis is the automation of ETL pipelines using advanced Artificial Intelligence techniques. It explores integrating these technologies into IBM InfoSphere DataStage through the automated generation of XML-based job templates, significantly simplifying and streamlining data integration processes. Custom scripts automate DataStage job configuration from natural language user requests, thus minimizing human effort and enhancing operational efficiency.

Chapter 2 provides an overview of the foundational concepts, methodologies and architectural evolutions that characterize modern data management and analytical processing

systems, highlighting their crucial role in supporting strategic decision-making within organizations. The discussion begins by introducing the fundamental concepts behind data Warehouse, exploring its significance as a structured and consolidated repository that integrates diverse sources of data to facilitate efficient analytics and informed decision-making.
After this general introduction the focus start onto the principal characteristics of data lake, data mart and databases, enhancing the benefit of data marts.
After discussing these aspects, the chapter progresses to explore in greater depth the structural architectures employed within data warehouse systems, focusing on Star Schema and Snowflake Schema. A comparison of these two schemas provides clarity regarding what are the advantages and trade-offs, enabling informed architectural choices aligned with specific business needs and analytical contexts.
Subsequently, the chapter expands the discussion to cover database processing methodologies, notably Online Transaction Processing (OLTP) and Online Analytical Processing (OLAP). Here, is explored the contrasting characteristics, design objectives and typical applications of these database systems.The integration and evolution of OLAP systems are further discussed, particularly emphasizing their increasing role in big data analytics. Recent developments such as the adoption of cloud-based deployments and the incorporation of machine learning algorithms have significantly expanded their analytical capabilities, enabling dynamic scaling and more advanced predictive insights.
In essence, this chapter outlines the evolution and interplay among various approaches to data warehousing, storage architectures and database management methodologies, providing readers with a clear understanding of how these technological advancements contribute to enhancing organizational decision-making and analytical processes. The chapter aims to equip readers with the knowledge required to navigate and implement effective data warehousing solutions.

In Chapter 3, the thesis provides an in-depth analysis of the MMCONS Framework, exploring how data is systematically collected, processed, validated and ultimately published to effectively support business intelligence and analytical operations. The chapter will guide the reader through each essential stage of this structured data-management pipeline, highlighting the critical tasks performed at each step and explaining their importance for the integrity, reliability and usability of the final analytical outcomes.
Initially is presented a detailed overview of ETL (Extract, Transform, Load) processes, which form the backbone of the MMCONS Framework. These processes are fundamental for ensuring data integrity, consistency and readiness for downstream analysis. The ETL pipeline is divided into three fundamental phases: Extraction, Transformation and Loading.
Subsequently, the chapter introduces and details the architecture and stages of the MMCONS Framework, a systematic approach for managing complex data workflows. This framework is designed to streamline and automate the processing of business-critical data, facilitating rapid, informed decision-making. Each stage of the MMCONS framework—ranging from initial data staging (Level L0 - Staging) through validation and transformation (Level L1 - Core Processing) up to the final data preparation and publication

(Level L2 - Publishing)—will be examined in detail, highlighting its specific role and functionalities.
Specifically, the Level L0 (Staging) represents the preliminary step where raw data from multiple heterogeneous sources, such as operational databases or external file feeds, is ingested into staging tables without undergoing any transformations.
After, the focus shifts into the core of data quality management within the Level L1 - Core Processing. This intermediate phase represents a cornerstone of the MMCONS framework, wherein the raw data loaded into staging is thoroughly validated, cleaned and transformed. Emphasis will be placed on essential validation mechanisms, including referential integrity checks and record validation rules.
The next stage, Level L2 - Publishing, represents the culmination of the MMCONS pipeline. In this final phase, data is organized according to thematic areas relevant to specific business processes, typically structured into Fact Tables and Dimension Tables to effectively support analytical operations. Here, the chapter will discuss the strategic considerations behind selecting between two common data warehouse architectures: Star Schema and Snowflake Schema.
The chapter wants to highlighting how the MMCONS framework exemplifies modern best practices in data management, striking a balance between simplicity, performance and accuracy. By systematically detailing each stage—from data acquisition to its final availability for analytical queries—the chapter underscores the critical role of structured frameworks in modern enterprises.

Chapter 4 introduces the architecture and core functionalities of IBM DataStage, highlighting its role in the context of ETL processes within data warehouse environments. The chapter presents the fundamental aspects of DataStage, emphasizing the functionalities and interconnections of its key components, which collectively ensure efficient data extraction, transformation and loading (ETL) operations.
The first part of this chapter outlines the main components of DataStage, detailing their specific roles within ETL operations. The role of the DataStage Manager is explained, emphasizing its critical function in managing project metadata and maintaining the integrity and consistency of the metadata repository. Next, the chapter addresses the DataStage Director, highlighting its operational and administrative capabilities, including job execution, real-time monitoring, logging, scheduling and alerting. Additionally, the DataStage Administrator component is examined, detailing its pivotal role in the configuration, optimization and security of the ETL environment.
The key elements discussed include the DataStage Designer, which provides an intuitive graphical interface to define and visualize data processing workflows through interconnected stages and links. This component is crucial because it serves as the primary tool for creating, visualizing and managing ETL jobs. The DataStage Designer utilizes visual programming through 'stages' and 'links', facilitating the clear definition and seamless management of complex data workflows.
Moreover, the chapter explores the different job types supported by DataStage, specifically distinguishing between Parallel jobs and Sequence jobs.
Finally, the chapter underlines the critical relevance of DataStage as an enabler of robust

data governance and strategic decision-making. The comprehensive understanding presented in this chapter serves to emphasize DataStage's importance in modern business intelligence and strategic decision-making environments, highlighting its role as a cornerstone in the effective management and integration of enterprise data.

Chapter 5 aims to illustrate the evolution of Natural Language Processing (NLP), starting from its foundational methods and progressing toward contemporary models and how NLP techniques transitioned from early statistical model to sophisticated neural architectures, emphasizing developments and key technologies that have revolutionized language understanding and generation.
Initially the chapter explores classical Statistical Language Model. These methods, which include techniques such as N-grams, represent language probabilistically by estimating the likelihood of word sequences based on statistical patterns found in training corpora. The limitations of purely frequency-based statistical approaches laid the groundwork for more semantically-rich methods, as for example Word Embeddings. Consequently, is discussed the advent and widespread adoption of Word Embeddings, particularly models such as Word2Vec which capture semantic and syntactic relationships, enabling NLP models to perform analogical reasoning and semantic similarity tasks with unprecedented accuracy. The methodologies behind word embeddings are explored, highlighting the context-based learning process (CBOW and Skip-gram in Word2Vec). However, traditional embeddings still had limitations, notably their inability to effectively handle out-of-vocabulary (OOV) words and their sensitivity to morphological variations. To address these challenges, attention is given to FastText, an advanced word embedding method developed by Facebook AI Research, this approach effectively handles OOV words, rare terms and morphological variations, significantly enhancing NLP tasks involving diverse linguistic forms and morphologically rich languages.
After examining these fundamental embedding methods, the focus shifts towards neural network-based sequential models, highlighting the critical role played by Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) architectures.
After this discussion, is introduced the transformative innovation of the Transformer architecture. The heart of the Transformer—its Multi-Head Attention (MHA) mechanism—is discussed in-depth, demonstrating its ability to concurrently capture multiple types of relationships across various tokens in sequences, significantly enhancing the representation and understanding of linguistic contexts. The exploration of the Transformer is further expanded by describing its Encoder-Decoder structure, which effectively facilitates complex tasks like language translation and text generation.
Following this, the focus shifts on two prominent Transformer-based models—BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-Trained Transformer)—detailing their distinct architectures, training strategies and application scopes.
Lastly, this analysis delves into the specialized area of Question-Answering (QA) systems, elucidating how contemporary NLP models, particularly Transformers, contribute to the effective design and deployment of QA solutions. It is outlined the operational components of typical QA architectures: question processing, information retrieval and answer extraction/generation. Moreover, is distinguished between extractive approaches

(predominantly BERT-based), which select precise answer spans from provided texts and generative approaches (typically GPT-based), capable of synthesizing new, contextually coherent answers.
Through this structured narrative, the chapter provides a holistic understanding of NLP's remarkable evolution, clearly illustrating how foundational statistical models and embedding methodologies have naturally progressed towards sophisticated neural network architectures.

Chapter 6 delves into the innovative core of this research: the automated creation of ETL jobs through dynamic XML generation using advanced AI techniques. This chapter outlines the developed methodology for translating natural language user requests into XML job configurations via semantic embedding techniques and generative models, thoroughly discussing implementation specifics, challenges encountered and practical solutions adopted during the project.
The chapter 6 presents the results derived from the experimentation and evaluation of the various techniques used within the developed pipeline, focusing particularly on the use of text embedding models and advanced prompting techniques with generative models. Specifically, a comparative analysis of the various text embedding techniques (Word2Vec, FastText and BERT) is initially provided, which are used to retrieve similar queries within a database. The performance of these techniques has been evaluated through standard metrics such as Precision@K, Recall@K and Top-1 Accuracy, with the aim of identifying the most effective approach in semantically representing user requests.
Subsequently, the chapter addresses the analysis of various prompting techniques applied to Gemini generative models, used to automatically generate XML files from user textual requests. The techniques explored include different simple approaches as Zero-Shot, Few-Shot, Chain-of-Thought, Task Decomposition and Iterative Refinement, both alone and combined . The results clearly highlighted how combined technique as Few-Shot with Iterative Refinement produce higher accuracy results with respect to simpler technique like Zero-Shot or Few-Shot alone.
In conclusion, the results obtained confirm the effectiveness of advanced strategies based on customized embeddings and iterative prompting with limited examples, highlighting how such techniques enable optimal performance in the context of ETL process automation through generative models. These results provide an important foundation for future investigations aimed at further refinements of the techniques and models used.

Finally, Chapter 7 presents the conclusions and suggests future research improvements, outlining possible developments to further enhance ETL process automation. It discusses how these advancements could provide increased flexibility, scalability and effectiveness in responding to evolving data requirements and business opportunities.

Chapter 2

State of the Art

2.1 Introduction to data warehouse

A Data Warehouse is a centralized repository that consolidates structured historical data from various sources. This data, originating from business activities, external entities, application outputs and log files, undergoes a transformation process to become uniform information. This refined data is then used by decision makers through Business Intelligence tools and other analytical applications to drive strategic initiatives.
There are several key attributes that differentiate data warehouses from other decision-support systems[24]:

1. **Subject-Oriented:** data within the data warehouse are organized by subject, making it easier to provide comprehensive information related to specific areas or fields.

2. **Integrated:** data warehouses combine data from different sources which often differ in format and encoding, ensuring that the information is consistent and consolidated.

3. **Time-Variant:** unlike operational systems that manage real-time data, a data warehouse stores data over an extended period. This characteristic supports the analysis of trends and the evolution of data over time, providing a historical perspective.

4. **Non-Volatile:** once entered into the data warehouse, the data become static. It does not undergo further updates or deletions, thus preserving its integrity over time.

It is essential to differentiate between the various systems used to manage company information resources. In next section are explored the distinct characteristics, needs and purposes of data lakes, data marts and data warehouses.

2.1.1 Data lake

Data Lakes are scalable repositories that store large quantities of data in structured and unstructured form. Due to the unrefined nature of the data, utilizing data lakes necessitates more advanced and dynamic analysis technologies compared to those that are used in data warehouses. These systems are designed to handle the complexity and scale of raw data, enabling flexible data processing and analysis strategies. In addition to their capacity to manage vast amounts of raw data, data lakes can significantly augment the capabilities of a company by leveraging the potential of newer and more diverse data types. This adaptability allows organizations to explore and extract value from data that was previously inaccessible or underutilized due to technological constraints. [25]

Furthermore, data lakes can contribute to the operational efficiency of legacy systems. By offloading data processing tasks from older and more rigid systems to more agile and scalable environments, companies can extend the life and performance of their existing IT infrastructure. This transition not only reduces the strain on legacy systems, but also allows them to operate more efficiently, focusing on their core functionalities while data lake handles the heavy lifting of data processing and analytics.

This dual capability makes data lakes an indispensable component of modern data architecture offering both, enhanced data analysis opportunities and practical solution to improve system efficiency.

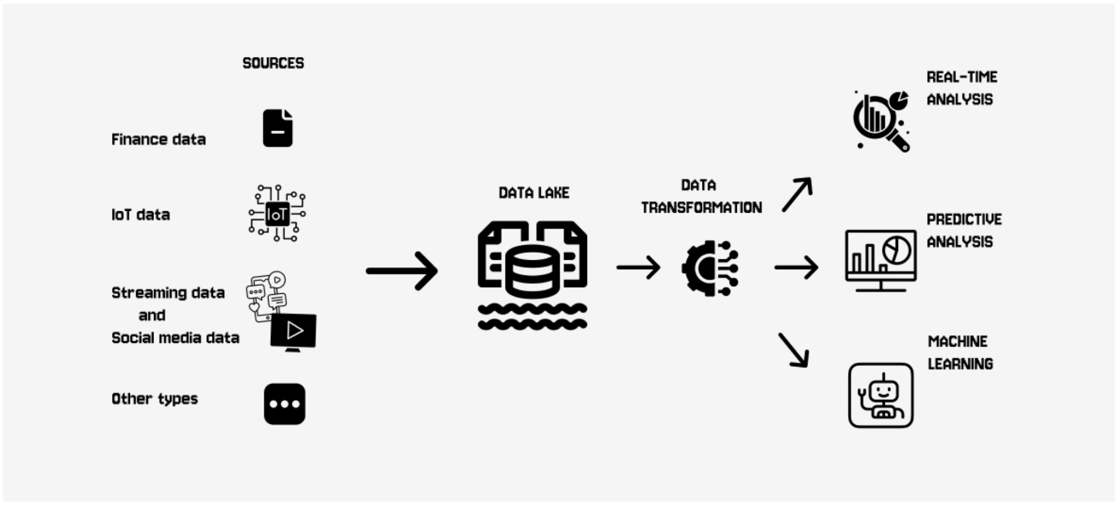


Figure 2.1. Organization and operation of Data Lake system

2.1.2 Data mart

Data Marts and data warehouses share structural similarities since both environments store consolidated and analyzed-ready data. However, data marts differs in their focus on meeting needs of individual segments or groups. Typically, data marts serve as specialized subset of data warehouses , each customized to the specific needs of distinct business units, such as finance or marketing.[1]
data marts provide several benefits:

- **Increased query performance**: with the fact that they have to manage smaller data volumes, data marts can offer rapid processing speeds, significantly enhancing query response times.
- **Enhanced security measures**: Data Marts restrict the access to data to only some stakeholders within specific departments, enhancing data security and enabling precise control over sensitive data.
- **Reduced complexity**: focusing on a singular business function reduces the complexity, in order to simplifies maintenance and administrator efforts

On the other hand, data marts have also some limitations:

- **Restricted scope**: given that they are limited to specific departmental data, data marts may not have all the business information that a larger data warehouse provides.
- **Potential for data redundancy**: there is the risk of overlapping data among multiple data marts, which can generate inefficiencies and increased storage requirements.

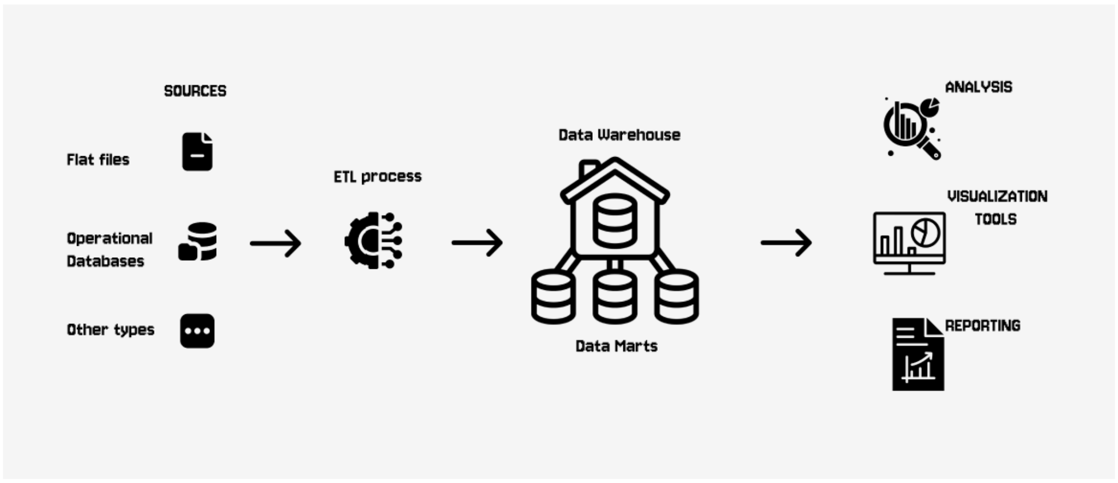


Figure 2.2. Organization of data warehouse system in the field of Business intelligence

Databases are collections of information arranged to have efficient storing, accessing and retrieval. Most databases have one purpose and have high speed reads and writes optimized for a single application. Databases actually can be in numerous forms such as XML, CSV and Excel spreadsheets. Databases differ in purpose and structure from data warehouses. They are useful to support immediate operational needs, while Data Warehouses work to facilitate strategic decision making through a consolidated view of business information.

As shown in picture 2.2, transactional databases act as standalone sources of information. Information in such and other sources is collected and cleaned out through ETL processes and they are stored in the data warehouse. While transactional databases utilize entity-relationship models and normalization for database schema, these methodologies are used less in data warehouses, whose operations rely on other architectural frameworks for preparing information for analysis and reporting.

2.2 The architectures of data warehouse: star schema vs snowflake

2.2.1 Star schema

The star schema is the widely used dimensional model when dealing with the creation of data warehouses and data marts due to its simple and efficient structure for querying large datasets. This schema is optimized for query performance by minimizing complexity through fewer joins.

As suggested by the name, the star schema has a structure where a central fact table is directly connected to various dimension tables, forming a star-like shape. The fact table contains quantitative data (facts) related to business processes events, or conditions and foreign keys that reference the surrounding dimension tables. Dimension tables, on the other hand, contain descriptive attributes that provide context for facts in the fact table. Primary keys in the fact table, which are viewed as a composite keys derive from foreign keys that connect the fact table to the dimension tables and foreign keys, which correspond to the primary keys of the dimensions, allow dimension tables to be linked to the fact tables. [13]

In this type of schema, there are normalized fact tables; instead the dimension tables are denormalized. This approach offers several advantages, including simplified query processing due to fewer joins and improved queries performance, which is very helpful for environments where speed is crucial. Conceptual modeling is a fundamental step when designing star schemas, ensuring data integrity, consistency and efficient data loading. [24]

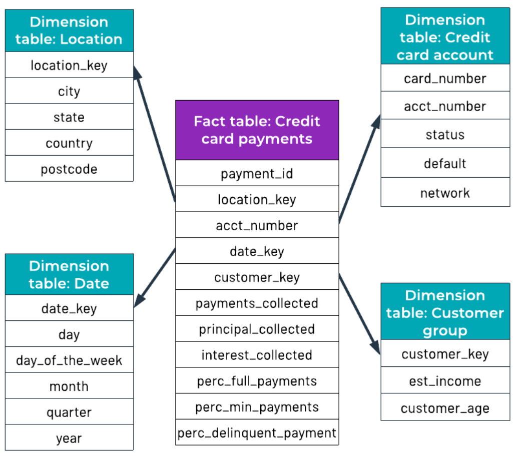


Figure 2.3. Example of star schema [10]

2.2.2 Snowflake schema

The snowflake schema is an extension of the star schema, it involves additional normalization of dimension tables, which are broken down into smaller related tables. This subdivision gives a reduction of data redundancy. Here the key management is slightly different with respect to the star schema, primary keys are present in each dimension and sub-dimension table, which is critical not only for ensuring data integrity but also for linking data across the schema efficiently. On the other hand the foreign keys play a key role in this schema , these keys not only connect the dimension tables to the central fact table but they also link sub-dimension tables to their respective parent dimension tables.

This approach has the advantages of higher data integrity and reduces redundancy through the operation of normalization. The disadvantages of this approach are the increased complexity in query processing due to the additional joins required and the higher maintenance demands due to the complex data structure with respect to the star schema. Snowflakes schema are often used for financial analysis or customer relationship management systems. Organizing detailed hierarchies and saving storage space are more important than query speed in these cases. [26][24]

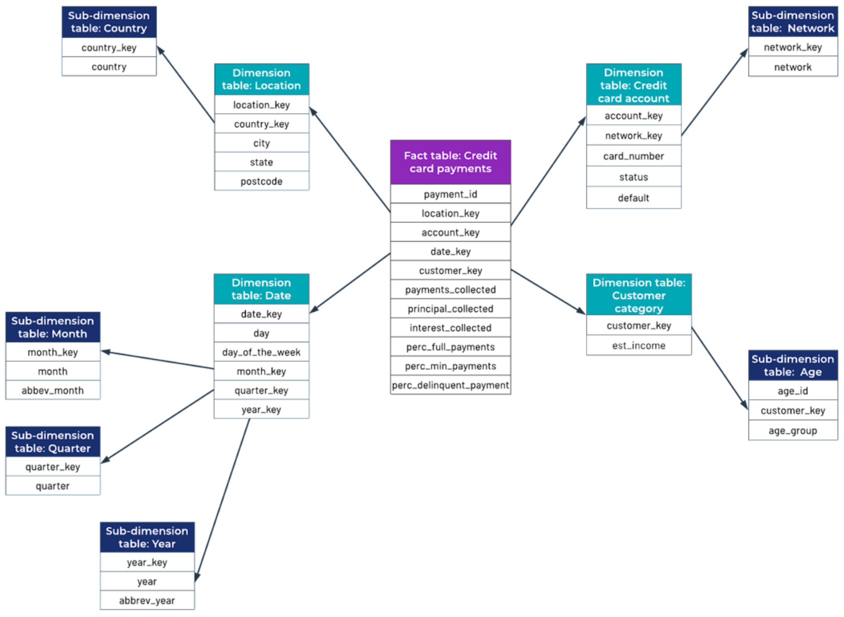


Figure 2.4. Example of snowflake schema [10]

In table 2.1 it is possible to see an highlighting of the key differences between a snowflake and star schema [27]:

Feature	**Star Schema**	**Snowflake Schema**
Architecture	Fact table with denormalized dimension tables	Fact table with normalized dimension tables
Complexity	Simpler to understand and design	More complex due to normalization and additional sub-dimensions
Normalization	Denormalized	Normalized
Performance	Suitable for simple data structures	Suitable for complex data relationships
Query Maintenance	Easier maintenance	More challenging maintenance
Storage	Requires more storage due to data redundancy	Requires less storage due to normalized structure
Data Redundancy	Higher redundancy	Lower redundancy
Query Performance	Faster queries due to simpler structure	Slower queries because of additional joins
Ease of Maintenance	Easier to design and maintain	More complex to design and maintain

Table 2.1. Star Schema vs. Snowflake Schema: A Detailed Comparison

2.3 Database: OLAP vs OLTP

In the world of data management, databases are broadly categorized into two types based on their primary utilization : OLTP (Online Transaction Processing) and OLAP (Online Analytical Processing).

2.3.1 OLTP

OLTP databases are specifically built to efficiently manage applications that rely on frequent transactions. These type of databases are known for transactional operations and play a critical role in support a business's day-to-day activities. Common tasks include adding, updating, or removing some data, ensuring in the mean time , real-time speed and accuracy.
Below it is possible to see some key features:

- **High concurrency:** OLTP systems are designed to accommodate thousands of users and transaction happening at the same time while maintaining consistent speed and reliability.
- **Short transactions:** transactions in OLTP are fast and usually involve few records to be efficient.

- **Data integrity:** mechanisms like locking and rollback are used to safeguard transactions and ensure data consistency throughout the process.
- **Normalized database schema:** OLTP databases typically employ a highly normalized schema, reducing redundancy and optimizing storage usage. [9]

In the contemporary tech landscape, OLTP systems are increasingly incorporating in-memory computing capabilities, which dramatically speed up data processing times and enable real-time analytics within transactional databases. This integration is pivotal for applications requiring instant decision-making capabilities, such as financial trading platforms an online retail transaction system.[8]

2.3.2 OLAP

On the other hand OLAP databases focus on enabling complex data analysis and queries over heavy datasets. They are essential for business in order to generate insights, produce reports and support strategic decision-making. The OLAP's key features are:

- **Multidimensional analysis:** OLAP data are structured in cubes , making it easier to perform analysis in multiple dimensions (e.g. time, location, product)
- **Handling large data volumes:** these type of databases can store and process large amounts of historical data for analysis purposes.
- **Complex queries:** OLAP supports advanced queries to cover trends, relationship and patterns in data.
- **Denormalized schema:** unlike OLTP systems, OLAP databases often use denormalized structures, such as star or snowflake schemas, in order to optimize performances during analytical tasks. [9]

The evolving role of OLAP in big data analytics is marked by its integration with machine learning algorithm to further refine data analysis processes and predictive modeling. Additionally, the deployment of OLAP systems on cloud platforms allow organizations to scale resources dynamically based on data volumes and computational needs, which is crucial for handling peak loads during critical business periods.

1. **Machine learning integration:** the enhancement of OLAP systems through the integration of machine learning algorithms can significantly improve predictive analytics. These systems can handle large volumes of data and apply machine learning models to perform complex predictions, thereby enhancing decision-making processes and enabling more accurate forecast.[28]
2. **Cloud deployment:** the deployment of OLAP systems in cloud offer scalability and flexibility, two crucial elements in order to managing large datasets and computational demands. This type of approach, called cloud-based approach, supports dynamic resource scaling, which is essential for dealing with variable workloads and peak data processing periods. Cloud environments also facilitate easier integration

with other data services and applications, enhancing the overall analytical capabilities of OLAP systems. [22]

In table 2.2 it is possible to see differences in the use and application of OLAP and OLTP databases.

Feature	**OLAP (Online Analytical Processing)**	**OLTP (Online Transaction Processing)**
Purpose	Used for complex analytical and decision-making queries	Used for day-to-day transactional processing
Data Structure	Historical and aggregated data for analysis	Current, real-time, detailed transactional data
Query Type	Complex queries for reporting, data mining and analytics	Simple queries focused on CRUD (Create, Read, Update, Delete) operations
Performance	Optimized for read-heavy workloads	Optimized for high-speed transactional processing
Data Redundancy	Often contains redundant data for performance optimization (denormalization)	Highly normalized to avoid redundancy and ensure consistency
Storage Requirements	Requires large storage due to historical data	Requires minimal storage since it only retains current transactions
Users	Used by business analysts, data scientists and executives	Used by front-end users, clerks and application programs
Backup and Recovery	Periodic backups, not time-sensitive	Critical backups required frequently due to real-time operations
Transactions	Fewer but complex transactions involving large datasets	Large number of simple transactions performed frequently
Example Systems	Data Warehouses, Business Intelligence tools (e.g., SAP BW, Google BigQuery)	Banking systems, e-commerce platforms, reservation systems

Table 2.2. OLAP vs. OLTP: A Comparative Analysis

Chapter 3

Introduction to ETL (Extract, Transform, Load) processes

Nowadays, companies need data to know what happens internally or externally, take decisions and maintain competitiveness on the market. Data are necessary but not sufficient. What truly matters is the quality of the data and, more importantly, the ability to utilize and derive value from it. [4]

ETL which means Extract, Transform, Load, is a three phase data integration process which allows making data available for analysis and all activities in the context of business intelligence, consolidating data from various source systems into a data warehouse, data lake or another target system to enhance data accessibility. ETL pipelines prepares data for analysts and decision makers, saving time and reducing errors associated with manual data processing. How said before, ETL pipelines is composed by three phases:

- **Extract**
- **Transform**
- **Load**

3.0.1 ETL pipelines - Extract

The first phase of the ETL process is the extraction stage, during it, the objective is to retrieve data from multiple source systems. The method used for extraction can be different depending on how the data is available at the source. It could require SQL queries to extract subsets from relational databases, API to extract data from web application and FTP protocols for flat files.[2] This process could be made in two ways:

- Full Load: all data from the source is transferred to the data warehouse of destination after transformation. This is typically done the first time data is loaded from the sorce to the target, in this case a data warehouse.[4]

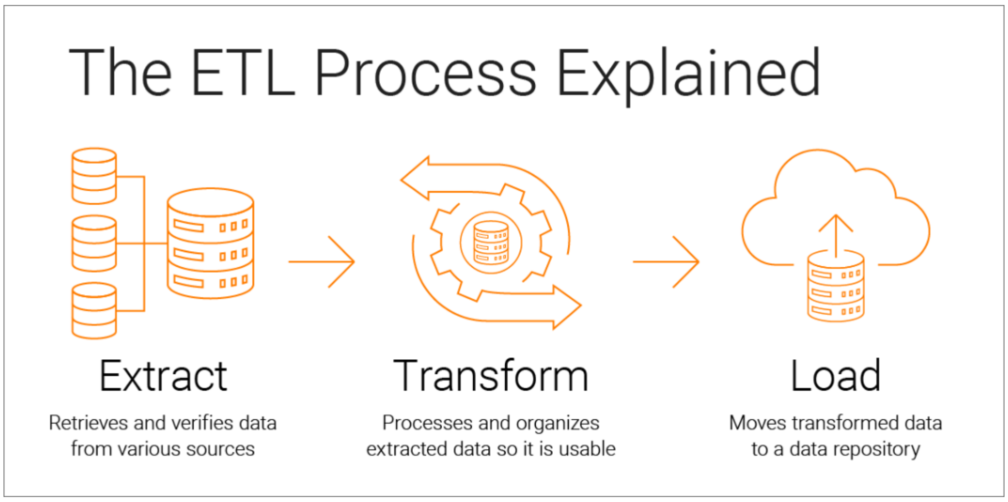

The ETL Process Explained
Extract
Retrieves and verifies data from various sources
Transform
Processes and organizes extracted data so it is usable
Load
Moves transformed data to a data repository

Figure 3.1. ETL Schema [2]

- Incremental Load: in this case only the records added after the last extraction are loaded. This can be done in streaming for low data volumes, allowing continuous data flow monitoring and processing. For large data volumes changes are collected and transferred in batches.[4]

3.0.2 ETL pipelines - Transform

After the extraction, there is the transformation phase , the most important phase of this pipeline, where data are processed with rigorous checks to ensure its values, schema, data type , structure and absence of error. This ensures the integrity of data , which is crucial for reliable analytics and decision-making. Various can be the techniques to transform data :

- **Aggregation:** data is aggregated to provide a full view, this is useful for analysis and reporting, in order to aggregate data can be done sums,averages or statistical measures
- **Data masking and encryption:** process to encrypt or masked sensitive information to protect privacy
- **Normalization and standardization:** this type of transformation ensure data respect quality standards helping in mantaining consistency.
- **Data formatting:** consist in convert data in required format based on the need of the target system.
- **Data merging and integration:** it consists in combining various source to create a unified view.

These transformations typically are done in a STAGING AREA . This area helps in isolating processing tasks from the operational system , improve performance and reduce the impact of transformation process. [2]
Implementing a well-designed data transformation strategy is very important for any company that aims to make data-driven decisions. It not only safeguards data integrity but also increase agility of business operations, ensuring business can quickly adapt to changes and new opportunities.

3.0.3 ETL pipelines - Load

This phase is the final step in the ETL process where the data transformed in the previous phase now are transferred to the destination. This can be an on-premise database, a cloud-based data warehouse, a data hub or a data lake , it depends on the organization rule and needs. A successful loading process starts with data mapping, which consists in defining how each data element from the source system aligns with a specific field in target system. Data loading techniques are different based on the use case:

- **Bulk Load:** designed for large datasets, this approach divides data in batches , reducing load time and optimizing transfer performance.
- **Incremental Load:** this method is designed for situations of frequent data updates, it transfers only the modified or new records, ensuring real-time freshness .
- **Full Load:** this method replaces the entire target dataset with the latest version , using all data. Useful for scenarios which need a complete synchronization.

3.1 MMCONS Framework

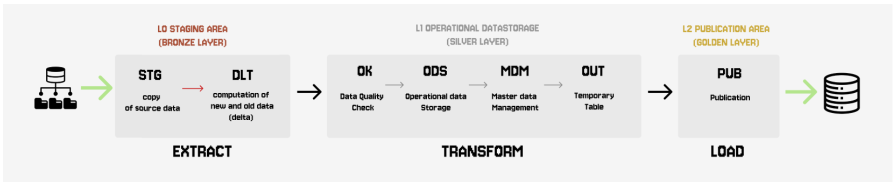


Figure 3.2. MMCONS Framework

A data pipeline is a sequence of steps that transform and manipulate data using various components. Each component processes data from the previous step and send it to the next one , creating a seamless automated data management flow. This framework typically starts by receiving data and culminates in a data repository where the processed data is stored in order to be used in a Data Warehouse. A more deep analysis will be conducted on the data flow framework implemented by the company Mediamente Consulting , focusing on its operational characteristics and functionalities. As after is possible to see it is composed by three main level : Level L0, Level L1, Level L2

3.1.1 Level L0 - Staging area

The first step in this data process is the staging area, which holds an exact copy of the raw data coming from various sources, like relational databases or customer-provided files. Since data is copied as it is, it might contains errors or be incomplete, but that is acceptable, data cleaning is not done at this point. Data is ingested in scheduled batches, with each batch (identified by a JOBID) handling one table or file at a time. During ingestion, three types of tables are used: delta tables, staging tables and error tables.

DLT (Delta) tables are used to store only the new or changed data coming from source systems. For every source table, there is a matching DLT table with the same structure. New data can be captured in three main ways:

- **CDC (Change Data Capture):** this method tracks just the changes since the last load, making it ideal for large tables because it avoids reloading everything. The data is ready to be processed right away.
- **MINUS:** if CDC is not available the system compares the latest version of the data with the previous one using a MINUS operation, however it requires temporarily storing both versions in staging tables.
- **FULL:** the entire table or file is reloaded each time, this is fine for small datasets or ones that do not change much. It is also used when is loaded a new source for the first time.

To protect against data loss or errors during the ETL process, it is important to keep a history of what was loaded before. This can be done either by saving the history directly in the DLT table (organized by JOBID), or by using separate history tables (called DLT_HIS) that also keep track of each load by JOBID. Thanks to this setup, often DLT tables do not need to store everything and so they can be quickly refreshed using a simple truncate and insert, which speeds things up.

STG (Staging) tables are used when data is loaded in full load in order to figure out what have changed compared to the previous load, so that only the differences are passed in the Delta (DLT) tables. It works as follow:

- The system compares the latest version of the data with the previous one using two steps:
  1. A **POSITIVE MINUS** finds the new or updated records.
  2. A **NEGATIVE MINUS** finds the deleted records.
- The results from both comparisons are then combined into one final list using a UNION, which is sent to the DLT table.

  To keep track of the type of change (insert, update, or delete), the DLT table includes an extra field which is often called *FLAG* that marks each record accordingly.

3.1.2 Level L1 - Transformation area

The subsequent, more critical phase of the ETL process involves transforming the staged data from mere replication of source structures to a format that integrates and reshapes the information for analytical processing. This phase includes normalization, integration and the generation and resolution of internal keys. Central to this stage is the data quality check, where data are validated through different mechanisms to ensure its accuracy and consistency, these type of checks are done from data which come from the DLT tables in the OK tables and consist of:

- **Referential integrity checks:** ensure that relationships across tables are maintained, using foreign keys to join tables correctly.
- **Record validation rules:** validate records against different criteria, such as non-null constraints and attribute-based checks , to ensure that all data meet the required standards.
- **Business rules**: apply specific business logic to discover suspicious data values, involving comparisons or methods to find outliers or errors.

After these checks, data which are unsuited for reconciliation, are 'sent' to tables used for the error management, the **ERR tables**. These error tables are present also in the layer L0, but they are analyzed in the layer L1 in order to store during the data quality checks, those records that fail such rules explained before. These records are temporarily excluded but reused in future processing cycles once the issues are resolved. ERR tables extend the original schema with error descriptions and technical fields from DLT tables. A retention period is set to automatically delete old records and prevent system overload.

In this layer are present also the **ODS(operational data store) tables** which represent the standard implementation of the L1 layer. They provide a centralized, normalized data model that meets the company's requirements and offer a consolidated and historic version of source data ready for integration. ODS tables are updated incrementally using MERGE statements and a primary key is required to enable record updates. Due to frequent updates, especially during multiple iterations in one load process, these tables are not compressed.

Furthermore other key structures in this layer are the **MDM (Master Data Management) tables** which play two key roles in the ETL process: integrating data from multiple sources and enriching it. They merge data from different ODS tables, collecting attributes into a central, unified structure, in order to avoid redundancy these tables only integrate new sources. In addition to integration, the MDM tables assign surrogate keys which are unique identifiers used instead of natural keys in order to improve query performance, simplify joins, reduce storage needs and to be more stable when attribute values change.

After the two steps of enrichment and integration of the data , it is transformed and organized to match the structure of the Data Warehouse, getting it ready for reporting and visualization. At this stage, the **OUT tables** come into play. They use surrogate keys to structure the data in a way that works well with visualization tools. Since data comes from different sources, it is common to find different ways of representing the same value. To avoid confusion or errors, a standard format is chosen for each value.

3.1.3 Level L2 - Publishing area

The final stage is needed to prepare and publish the data. This stage is divided in thematic area that align with specific business processes structured into:

- **Fact tables**: store quantitative metrics of business operations and are linked to dimension tables through foreign keys.
- **Dimension tables**: contain attributes which help aggregate the fact table data, in order to facilitate intelligence and reporting.

Data at this stage are modeled both as a Star Schema , with a central fact table connected to non normalized dimension tables and as a Snowflake Schema, where the dimension tables are normalized. Depending on the specific needs of the business and analytical performance considerations, it is possible to choice between these models, preferring models that minimize computational load during analysis. This framework outlines how data flows from initial assimilation to final analysis, ensuring data is accurately captured, processed and stored, ready for strategic business decision-making.

Chapter 4

IBM DataStage architecture

4.1 IBM Datastage overview

IBM DataStage offers a suite of tools which work together in order to facilitate the design, management, execution and administration of ETL processes. Each of the component has a different role , satisfying different aspects of the data integration workflow:

- **Datastage Designer**: it is the primary interface to build ETL jobs. It provides a graphical environment where the user can create data integration solutions by using various types of stages onto a canvas and configuring them to create complex ETL processes. Designer allows users to visually define data extraction, transformation and loading. This tool is essential for ETL developers as it supports the creation of both simple and highly complex transformations and integrations.
- **Datastage Manager**: it is used for managing metadata associated with Datastage projects. It acts as a repository management tool where all Datastage objects such as jobs, job sequences and other components are organized and maintained. The manager ensures that the metadata are consistent and accessible across projects and jobs, providing an essential function in managing the ETL job history.
- **Datastage Director**: it is used for validate, schedule and monitor jobs. It provides ETL administrators and operators with tools to execute and control jobs developeed in the Designer. With the Director the users are able to view job logs, schedule job runs , set alerts and monitor job-execution in real-time. It is important for maintaining operational health and efficiency of ETL workflows, ensuring that data loads and transformations continue as planned without errors or interruptions
- **Datastage Administrator**: it is responsible for the overall configuration and management of DataStage environment. This tool allows administrators to set up the properties of projects, configure global settings, manage user's roles and permissions and handle resource allocations. Administrator ensures that the environment is optimized for performances and security and that is compliant with organizational policies and data governance standards.

These components together create a robust framework for handling all aspects of ETL processes, from design and implementation to management. Each tool is specialized in different specific tasks within the data integration life cycle, making IBM datastage a very interesting solution for enterprise data management challenges. [7]

4.1.1 IBM Datastage Designer

Within the suite of IBM Datastage's tools, the Designer holds an important position for its direct engagement with the construction and visualization of ETL jobs. The Designer provides a sophisticated interface where the complexity of ETL processes is managed through an intuitive interface. It uses various components called 'stages' which are connected among each other by 'links', this in order to define flows of data through ETL processes. These stages can be view as modular elements of the software, where each stage performs a specific function such as extracting data, transforming data or loading data into a database. **Connectors** are specialized stages which allow the connection with various data sources and targets. They make easier the extraction and load of data by interfacing with different databases systems, as for example relational databases, flat files or big data systems. It is possible to find many type of stages which perform different functionality. The main types are:

- **Source stages**: enable the extraction of data
- **Transformation stages**: like transformer, aggregator and sort stages, which aggregate or modify data.
- **Target stages**: for loading data into a data warehouse or other storage systems.

These, depending on how they are connected to each other can perform various functions in order to process data at the various levels of the framework explained before.

There are two types of job which can be created in DataStage:

- **Parallel jobs**: they permit to the data transformation process to run in parallel using multiple processors, this enhances performances by distributing the workload across various resources, speeding up data processing tasks.
- **Sequence jobs**: they manage the execution of other jobs and activities in Datastage. These jobs allow the schedule and orchestration of multiple jobs, specifying the execution order and conditions for starting each job.

Through the use of IBM DataStage Designer, the enterprise achieves a refined control over data processing activities, allowing developers to construct highly efficient and customization ETL workflows. This tool's capabilities are very important to translate complex data integration requirements on executable and manageable tasks, also supporting robust data governance and strategic decision-making processes in business environments.

4.1.2 Parallelism and Partitioning

One of the key strengths of IBM DataStage lies in its ability to execute ETL jobs in parallel, significantly improving performance and scalability. DataStage supports multiple forms of parallelism:

- **Pipeline parallelism**: it allows different stages in a job, in order to process data simultaneously, passing results from one stage to the next without waiting for the entire dataset to complete.
- **Partition parallelism**: it splits datasets into partitions that are processed independently across multiple nodes or processors.
- **Data parallelism**: it distributes data subsets across different processing units that execute the same logic in parallel.

The use of partitioning strategies makes it possible to balance workload distribution and optimize data flow.

4.1.3 Metadata management and reusability

In automated ETL flows, consistency and maintainability are essential. DataStage promotes these qualities through metadata-driven design and reusable components.

- **Shared containers** encapsulate reusable logic or groups of stages, enabling standardized patterns to be reused across multiple jobs.
- **Parameter sets** allow for dynamic configuration of jobs at runtime, reducing the need to duplicate logic for different environments or scenarios.

These features contribute to reducing development time and simplify maintenance, which is particularly important in projects where ETL flows must adapt to evolving business requirements with minimal manual intervention.

4.1.4 Logging and error handling

Automated ETL systems require robust monitoring and error management to ensure reliability without human supervision. DataStage provides detailed logging capabilities through the Director tool, where each job execution is recorded with metadata such as timestamps, record counts, warnings and errors.

Additionally, DataStage allows the use of **reject links** to redirect records that fail validations and configurable **alert systems** to notify administrators in case of job failure or threshold violations.

These mechanisms ensure that the ETL process can self-monitor, isolate issues and continue processing when possible key aspects in an automated data pipeline.

Chapter 5

The revolution of language models: from statistical methods to transformers

5.1 Introduction

In the last years, the field of natural language processing (NLP) has experienced an important revolution, thanks to the emerging of the large language model (LLM). These model, as for example GPT-3,GPT-4, BERT and their successor , modify the way of processing natural language. Enabling applications from the machine translation to content generation, question answering and virtual assistants.
First LLMs were based on statistics approach which, also if they represent fundamental steps, they were limited in their ability to capture semantic and contextual complexities of language. A real turning point there was with the introduction of the embedding techniques as Word2Vec or GloVe which are able to represent words in vector , in order to be processed to more sophisticated neural networks capable of capture the subtle nuances of semantic and generalizing knowledge from large data sets.This was the first step towards creating systems capable of interpreting language in a more natural and contextualized way.
After, subsequently to the arrival of the transformer and the 'attention' mechanism, the models begin able to process complete sequence of words in parallel, managing the long-term dependencies present in the text. This innovative approach allowed us to increase the number of parameters and consequently, the ability of these models to learn and generalize in a surprising way. This evolution is driven by both incremental progresses and radical innovations that refined the standards of NLP.
This chapter wants to give an historical and technical panoramic on the evolution of the LLM, analyzing the principal evolution events, from the embedding models to transformers and question-answering systems which open the way to the actual pre-trained and fine-tuned models, giving also a particular attention to the future perspectives and the

open challenges in this field. These models not only represent a revolution from a technical point of view but they constitute a new paradigm for understanding and modeling the complexity of human language and also open up new perspectives for the future of digital communication and artificial intelligence. Below is possible to observe figure 5.1 which shows the various step from one of the first statistic model to the nowadays chat-gpt models.[32][19]

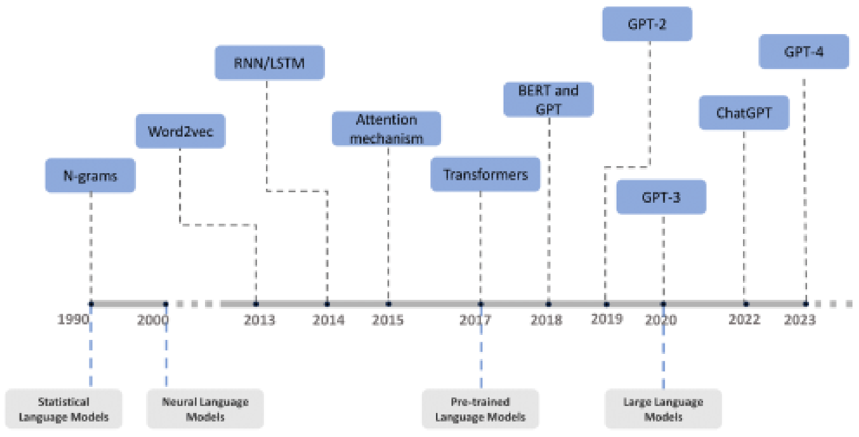


Figure 5.1. Development of Large Language Model

5.2 Origin of the natural language processing: statistics model

One of the first statistics model is the N-GRAM model which want to predict the next element in a sequence based on N previous elements. It is based on the principle of probability , in which the probability of a word or a sequence of words is determined by their frequency occurrences in a dataset. It divides the text in smaller units called 'n-gram' allowing the model to capture the relationships among the words. It is possible to have different type of models depending on the value assigned to n. For example in a 'bigram' model (n=2) the probability of a word is approximated as follows:

$$P(w_1, w_2, \ldots, w_N) \approx \prod_{i=1}^{N} P(w_i \mid w_{i-1})$$

Instead a 'trigram' model (n=3) considers the context of the two previous words

$$P(w_1, w_2, \ldots, w_N) \approx \prod_{i=1}^{N} P(w_i \mid w_{i-2}, w_{i-1})$$

This simplification reduces the computational complexity of the problem but introduces some important limitations . In fact, in order to obtain better context of the text, are needed higher values of n, but this lead to an higher computational complexity.[11]

5.3 Word embeddings

With the Word embedding techniques are intended a numeric representation of words in lower dimensional spaces capable of capturing semantic and syntactic information.
Their usage plays a very important role in natural language processing since they allow to represent a phrase with a numeric vector and so, the words with similar meanings would have similar representation. Large input vectors will result in a huge number of weights which will lead to an high computation required for training, with the word embeddings is possible to avoid this problem. The word embedding are used:

- as input to machine learning models; the process will result in :
  take the words–> Give a numeric representation–> Use in training or inference
- to represent any pattern in the corpus used to train them.

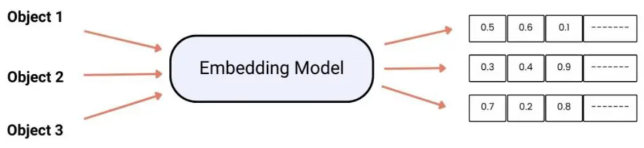


Figure 5.2. Word Embedding mechanism [3]

There are different approach for text representation:

5.3.1 Traditional approach

The traditional method involves the compilation of a list of distinct terms and the assignment of a unique integer value to each term. In this way a wide vocabulary results in a very large features size. Traditional method includes:

One-hot encoding: it is a simple method which represents words in the natural language processing , where every single word is represented by a unique vector and the dimension of the vector is equal to the dimension of the vocabulary. The vector has all the elements equal to zero except for the element corresponding to the index of the word in the vocabulary which is set to 1 . This type of encoding is a very simple and intuitive method to represent words, but it has some disadvantages lead to make it inefficient in certain application. In fact this method will produce high dimensional vectors which need high computation in calculus and memory , it represents each word as isolated from the context so it does not capture the semantic relationships among words and is limited to the vocabulary used during training.[3]

Bag of Words (BoW): it is a technique which represents a text as an unordered set of words, capturing the frequency of each word in the document creating a vector representation. This mode of work lead to lose some sequential and contextual information , making it inefficient for those application where the order of words is important as for example in the comprehensions of natural words. Furthermore it has a sparse representation with a lot of element equal to zero and this results in an inefficient computation and higher memory requisites when working with large datasets.

Term-Frequency Inverse-Document-Frequency (TF-IDF): it is a numerical statistic techniques that reflects the importance of a word in a document with respect to a collection of document called 'corpus'. Widely used in the natural language processing it is composed by :
Term Frequency TF which represents the frequency of appearance of a word in a documents so it can be calculated as:

$$\mathrm{TF}(t,d) = \frac{\text{Total number of word in document D } d}{\text{Total number of time the term } t \text{ appear in the document } d}.$$

Inverse Document Frequency measures the importance of a certain terms in a collection of documents, calculated as:

$$\mathrm{IDF}(t,D) = \log(\frac{\text{Total documents}}{\text{Number of document containing the term } t}),$$

and the final formula will combine TF and IDF in a single value

$$\mathrm{TFIDF}(t,d,D) = \mathrm{TF}(t,d) \times \mathrm{IDF}(t,D).$$

the higher is the score of TF-IDF for a term, the higher will be the importance of that term for that document with respect to the entire corpus. This results very useful in application as text mining, in the information retrieval and clustering of documents. Also this method introduce some limitations as for example the sensitivity to the document length, in fact long documents may have higher overall terms frequencies, potentially biasing TF-IDF towards long document.[6]

5.3.2 Neural approach

The main neural approach to generate word embedding is the Word2Vec,it is a popular technique in the NLP and it tends to represent words as continuous vector spaces. The main idea is that, words with similar meanings should have similar vector representation. In this method to a single word is assigned a single vector. It is possible to divide this method in:

Continuos Bag of Words (CBOW): it is a feedforward neural network architecture used in Word2Vec , its main objective is to predict a target word with respect to the context which consists in the words included in a given window, the target word will be in a center of this window context. [12]

Skip-Gram: with this architecture it can be said that it performs the inverse process of the previous explained Continuos Bag of Words. In fact in this architecture, the objective is to predict the context words which are around a target word given as input.

In figure 5.3 it is possible to see the key difference between CBOW and Skip-Gram, which is the direction of prediction.[12]

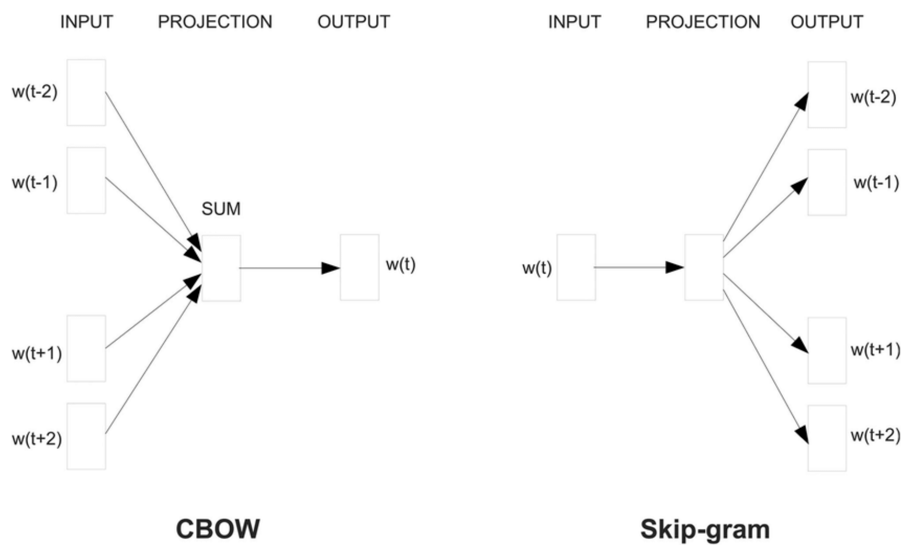


Figure 5.3. difference between CBOW and Skip-Gram

But there are also other differences such as:

performance with different word types: in fact , CBOW is particularly effective with frequent words , it captures syntactic information well. On the other hand, Skip-Gram excels in working with rare words and understanding semantic relationships. This make Skip-Gram particularly useful for application such as semantic similarity or text generation.

Computational efficiency: regarding this, CBOW is less resource intensive than Skip-Gram . Indeed CBOW aggregates context words in a single representation, simplifying calculations, while Skip-Gram requires more computations because need to predicts multiple context words from a single input word.

Use cases: both have their specific applications in Natural Language Processing. CBOW is choosen when training resources are limited or when the priorities is to capture syntactic informations. Skip-Gram indeed is widely used in applications which require semantic similarity, information retrieval and when understanding relationships between words is critical.

Even if Word2Vec is an algorithm used in a lot of applications, it suffers of some limitations:

- **Dependence on large amounts of data:** if it is used with a small corpus, the model is not able to capture the complexity of relationships.

- **Problem with homonymous and antonymic words:** it can have difficult with words semantically related but dissimilar that share similar contexts, such as synonyms and antonyms.

- **Inefficient management of Out Of Vocabulary (OOV) words:** in scenarios with an evolving vocabulary, the lack of certain words in the training set limits the model's adaptability.

5.4 Fast Text: A Word2Vec improvement:

Nevertheless , in order to solve problems of the Word2Vec, is introduced the FastText, a word embedding model which improves word embedding technique introducing the concept of representing the vector of the word as a sum of character n grams. It is a open library introduced by Facebook in order to deal with the lot amount of data generated by the users every days. It treats each word as compose of the sum of characters n-grams, which is the key difference with respect to the original Word2Vec.

For example, the word "kingdom" results in a sum of n-grams like this below:

['k','in','kin','king','kingd','kingdo','kingdom',...]

with this technique each word is represented as the combination (sum and average) of the n-gram components. The word vectors which are generated have the information about not only the word itself, but also about all the sub-word components. For instance in the example about "kingdom", one of the sub-component is "king", this enabling the model to establish semantic connection about words and capture also the meaning of suffixes and prefixes.

Additionally FastText generates better word embedding for rare or unseen words like the out of vocabulary (OOV) words. It could maintains its accuracy also without remove the stopwords, allowing for a simple pre-processing.

Due to its ability of leverage sub-word information, FastText can be particularly useful for morphologically rich languages such as Spanish, French or German. However, while it provides more refined word embeddings, it requires more memory if compared with Word2Vec, because it generates an high number of sub-words for each word. [5]

5.5 From RNN to Transformers: The Evolution of Deep Learning in NLP

5.5.1 Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM)

The Recurrent Neural Networks (RNNs) is a class of neural network built in order to elaborate sequential information. Differently from the feed-forward neural networks (which elaborate data in a single direction), Recurrent Neural Networks are equipped with recurrent connections which allow to 'memorize' or more precisely maintain the information about a past concept and use it during the elaboration of the next elements of the sequence.

A RNN is consisting of :

- **Input layers:** receive data inputs (text token, audio signal frame)

- **Hidden layers:** each containing an hidden state h_t which is updated by both current inputs and previous hidden state.

- **Output layers:** which produce the final result

The key aspect is the presence of an hidden state which is updated at each timestamp t. The formula for a standard RNN can be summarized as follows:

$$h_t = \sigma(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t + b_h)$$

where:

- h_t is the hidden state at time t.
- h_{t-1} is the hidden state from the previous time step.
- x_t is the input at time t.
- W_{hh} and W_{xh} are weight matrices.
- b_h is the bias term.
- σ is an activation function (e.g., tanh, ReLU).

The output at time t is often given by:

$$y_t = W_{hy} \cdot h_t + b_y$$

where W_{hy} is the projection from the hidden state to the output, and b_y is the corresponding bias term.

Recurrent Neural Networks are good for tasks where order and context are important, such as:

- **Natural Language Processing**
- **Speech recognition and Speech Synthesis**
- **Time Series Prediction**
- **Sequence Generation**

However, a problem of the standard RNNs is the long-term dependencies, in fact they often face with vanishing or exploding gradients during backpropagation, when have to deal with very long sequences.[17]

In order to handle this problem is introduced the LSTM (Long Short-Term Memory) networks.

The cells of this type of networks maintain an internal memory and use a system of gates to control how information flows through the time:

- **Input gate:** determines how much new information is stored in the internal state from the current input

- **Forget gate:** decides which parts of the internal state are discarded or not.

- **Output gate:** Controls how much of the internal cell state is exposed as the hidden state output for each time.

With this structure LSTM networks can preserve information for longer periods and so, they are better at capturing long-range dependencies. [17]

5.5.2 Transformers and Multi-Head Attention Mechanism

The transformer architecture is proposed for the first time by Vaswani in 'Attention is All You Need' [30] and it marks a turning point in the field of Natural Language processing (NLP) eliminating the reliance on sequential models such as RNNs and LSTMs, which had a significant limitations in handling long term dependencies and parallelizing computations. This leads to a drastic reduction in training times and allows the efficient management of very long sequences.
The heart of the Transformer architecture is the **Multi-Head Self-Attention Mechanism (MHA)**, an extension of the self-attention mechanism that allows to model the relationships between words in a sequence independently of their distance. The main advantage of this approach is the ability to capture both local and global dependencies without the need for sequential propagation of information [30]

5.5.3 Self-Attention Mechanism

Attention is an operating that given a set of input vectors, computes a similarity score between each pair of tokens based on their contextual relevance [20]. This mechanism is built around three fundamental matrices **Query (Q), Key (K) e Value (V)**. The input X (represented as an embedding matrix of size $n \times d_{\text{model}}$) is transformed into these three spaces through multiplications with weight matrices learned during training:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V \tag{5.1}$$

where W_Q, W_K, W_V are the projection of matrices of size $d_{\text{model}} \times d_k$.
The attention score is calculated by the normalized dot product between Q and K, followed by a softmax function to obtain a probability distribution:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{5.2}$$

The division by $\sqrt{d_k}$ serves to reduce the variance of the resulting values and stabilize the training process. The softmax function ensures that the attention weights are distributed in such a way as to give greater emphasis to the most relevant tokens. [18]
However, in some contexts, such as auto regressive text generation, it is necessary to restrict the model's ability to attend to future tokens. This is achieved by applying a masking mechanism that sets certain attention scores to negative infinity before applying the softmax function. This prevents the model from accessing information from future positions, ensuring a causal flow of information.

Furthermore, the original attention mechanism operates on a single representation space, limiting the diversity of relationships that can be captured. To enhance this, **Multi-Head Attention** is introduced [30]. Instead of applying a single attention function, multiple attention heads operate in parallel, each with its own projection matrices W_Q^i, W_K^i, W_V^i:

$$\text{head}_i = \text{Attention}(QW_Q^i, KW_K^i, VW_V^i) \tag{5.3}$$

The outputs of all heads are concatenated and projected back to the original space using a learned weight matrix W_O:

$$\text{MHA}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W_O \tag{5.4}$$

This mechanism allows the model to capture multiple types of relationships simultaneously, improving its ability to model complex linguistic structures.

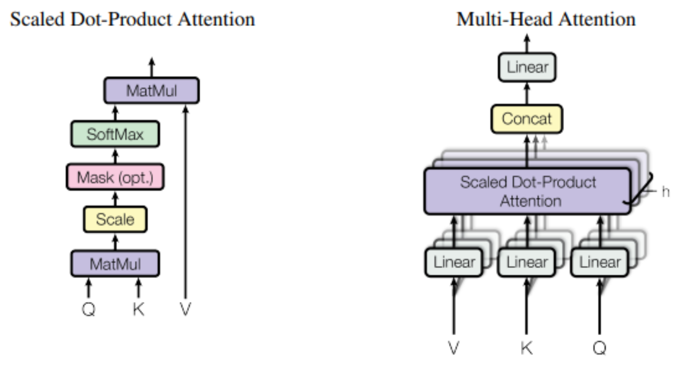


Figure 5.4. Left: Scaled Dot-Product Attention mechanism. Right: Multi-Head Attention.

5.5.4 Encoder-Decoder structure in transformer

The original transformer architecture, introduced by Vaswani et al. [30], consists of two main components: the **encoder** and the **decoder**. These components play complementary roles in the sequence processing process as is possible to see from the figure below 5.5.

- The **encoder** is responsible of encode the input sequence into an abstract representation, also called latent representation. It consists of a series of identical blocks stacked on top of each other, each of which contains two main sublayers: a multi-head attention mechanism (*Multi-Head Self-Attention*), followed by a feed-forward fully connected neural network (*feed-forward neural network*). Each sublayer uses *residual connections* and *layer normalization* to stabilize training and accelerate convergence. The final representation produced by the encoder contains rich contextual information and is used by the decoder to generate the final output.

- The **decoder** uses the representation produced by the encoder to generate a target sequence (e.g., a translated sentence). It is also composed of several similar blocks stacked vertically, each consisting of three sublayers: a first layer of masked self-attention that allows the decoder to see only the previous tokens in the generated sequence, a second layer of multi-head attention that considers the output coming from the encoder and finally a feed-forward neural network similar to that of the encoder. Masked attention is essential to ensure that during training the model cannot access future information and is forced to learn to generate coherent sequences step by step.

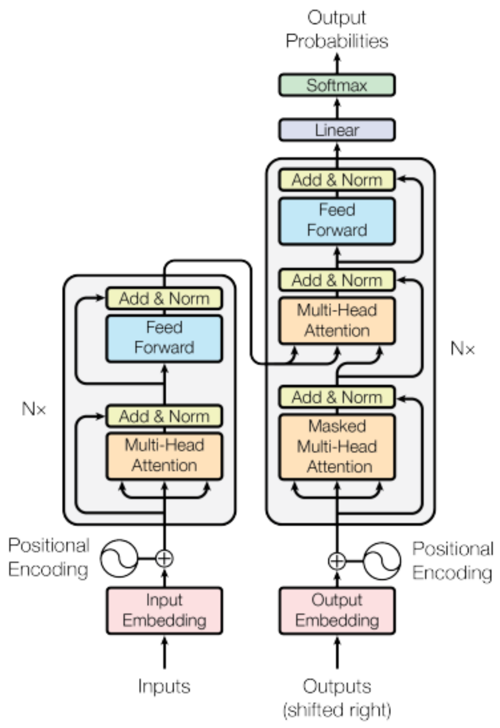


Figure 1: The Transformer - model architecture.

Figure 5.5. Transformer architecture. [30]

Practically, the decoder computes the final output based on the latent representations generated by the encoder and the tokens already generated:

This type of architecture is particularly effective in tasks such as machine translation and text generation, where it is essential to understand a source sequence and generate a syntactically and semantically coherent target sequence. In cases where it is wanted to use the transformer for tasks not related to sequential generation (e.g. text classification), it is often sufficient to use only the encoder component, as it is done in models such as **BERT** or **RoBERTa**.

$$\text{DecoderOutput} = \text{Decoder}\left(Y_{\text{input}}, \text{EncoderOutput}\right) \tag{5.5}$$

where Y_{input} represents the tokens previously generated by the decoder itself and EncoderOutput represents the encoded sequence.

5.5.5 Real application of transformer

The introduction of the Multi-Head Attention Mechanism transforms the paradigm of NLP models, offering richer and more parallelizable context modeling than traditional recurrent networks. Due to its efficiency and flexibility, Transformer becomes the standard for many deep learning applications and current research suggests that its impact will continue to grow in the coming years.
Applications of transformers are used in a wide range of real-world applications. Some notable examples include:

- **Automatic translation:** google translate has left its old LSTM-based model in favor of transformers, improving quality of translation and reducing processing times.
- **Search engines:** google has integrated the BERT model (a Transformer-based implementation) into its search engine to improve the understanding of user queries.
- **Virtual assistants and chatbots:** transformer-based models are the basis of Alexa, Siri and Google Assistant, improving natural language processing capabilities and making interactions more natural.
- **Biomedical data analysis:** transformer-based models are used in genomic data analysis and AI-assisted medical diagnosis. For example, some models are trained to analyze DNA sequences in order to recognize patterns associated with specific genetic diseases.
- **Content Synthesis and Generation:** models like GPT-3/ GPT-4 are used to generate articles, stories, code and even creative content like poetry and videos

5.5.6 Pre-Trained models: Bert & GPT

Pre-trained language models have revolutionized NLP by learning universal language representations, which can be fine-tuned for various tasks. Two of the most influential transformer-based models are **BERT**(Bidirectional Encoder Representations from Transformers) and **GPT** (Generative Pre-Trained Transformer). Both use the transformer architecture but with different designs and objectives, leading to distinct strengths.
In fact, while Bert is an encoder-only transformer that reads text bidirectionally, in contrast GPT is a decoder only transformer used in an auto regressive (unidirectional) way. So Bert's self-attention considers both left and right context of each word simultaneously, on the other hand GPT processes text left-to-right, where each token only attends to previous tokens. [16]. This key difference means BERT deeply encodes full sentence context while GPT generates text predicting next word.

BERT's bidirectional nature allows it to capture rich contextual dependencies, making it very effective for tasks like question answering and token classification where is crucial to understand the entire input. GPT's unidirectional (causal) modeling makes it naturally suited for language generation tasks, as it produces text one token after another in a coherent sequence. However, GPT's left-to-right constraint can be sub-optimal for tasks that require understanding context from both ends of a sentence. Conversely, BERT cannot natively generate free-form text because it lacks a mechanism to produce sequence output autoregressively; it encodes but does not decode text.

In figure 5.6 it is possible to see the difference in architecture.

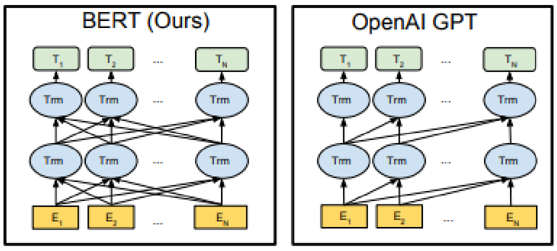


Figure 5.6. Difference between BERT(bidirectional transformer) and OpenAI GPT(Left to Right transformer)[16]

In summary, BERT's encoder architecture is optimized for understanding and extracting information from text (classification, retrieval, QA), since every input token's representation integrates all context. GPT's decoder architecture is optimized for generation – it excels at producing fluent and contextually consistent continuations of text (story generation, summarization) due to its auto regressive nature.

Regarding pre-training, it is done with different self-supervised learning tasks.
BERT uses **Masked Language Modeling (MLM)** and **Next Sentence Prediction**

(NSP). In MLM , random tokens in the input are masked and the model must predict them, forcing learning from both left and right context. NSP is a secondary task where the model learns to predict if one sentence follows another, improving BERT's handling of sentence relationships. [16]
On the other hand, **GPT** uses a **Causal Language Modeling (CLM)** objective, it learns to predict the next token in a sequence, given all previous tokens. This unidirectional training aligns with GPT's left-to-right architecture and directly trains it for text generation.
Furthermore the models also differ in the scale and nature of pre-training data. BERT is trained on a smaller but high quality corpus (BookCorpus ˜ 800M words + English Wikipedia ˜ 2,500M words)[16] instead GPT leverage web-scale data , the original GPT is trained in BookCorpus, GPT-2 is trained on 40+GB of WebText and GPT-3 (175 billion parameters) is trained on around 300 billion tokens from diverse sources (Common Crawl web data, books, Wikipedia, etc.), orders of magnitude more data than BERT. This massive scale gives GPT-3 with broad knowledge and the ability to perform many tasks with little or no fine-tuning.
Both BERT and GPT are fine-tuned for downstream tasks, but while BERT typically uses supervised fine-tuning, adding a small output layer to its pre-trained encoder for tasks like classification or span extraction, GPT often employs prompting or instruction tuning. Notably, GPT models have been further refined using Reinforcement Learning from Human Feedback (RLHF), a process that aligns the model's outputs with user intents, as seen in InstructGPT, ChatGPT and GPT-4. In contrast, BERT's fine-tuning remains primarily focused on understanding tasks and does not incorporate RLHF, reflecting its design for comprehension rather than free-form generation.
In practice, these models can complement each other in complex NLP systems. For example, one can use BERT to interpret or retrieve information (thanks to its strong comprehension of context) and then feed that information to GPT to generate a natural language response. Research in speech recognition has shown that combining bidirectional and unidirectional models (BERT with GPT) yields improved results.

5.5.7 Question-Answering system

A question-answering (QA) system is a software system designed to automatically answer question posed in natural language by understanding the query, retrieving relevant information and providing a concise answer.[14]. Such systems combine techniques from natural language processing and information retrieval to deliver precise results.
QA technology is highly valuable in real-world applications: for instance, search engines leverage QA components to deliver direct answers to user queries instead of just lists of documents and customer service platforms employ QA to automatically handle frequently asked questions, providing instant and consistent support. [21].
Furthermore knowledge-based AI assistants (e.g. voice-activated virtual assistants) rely on QA techniques to interpret user queries and produce helpful answers in a conversational manner, making QA a cornerstone of modern information-access systems.
The typical QA system has an architecture composed of three key components, each responsible for a stage in the process[23]:

- **Question Processing**: in this step the system analyzes the input question in order to understand what the user is asking. This involves natural language processing steps like parsing and keyword extraction to form a query, as well as identifying what is the focus and what is the expected response [23].
- **Information Retrieval (IR)**: in this step the system searches a large collection of documents or a knowledge base for relevant information that could contain the answer. Traditional IR methods are often used to fetch a set of candidate passages, but modern QA systems increasingly use vector-based semantic search powered by deep learning.[31]
- **Answer Processing**: this is the final stage, where the system examines the retrieved text and extracts or formulate the answer. The system may employ techniques such as named-entity recognition, answer type checking or a reading-comprehension model to select the most appropriate answer. Then the chosen answer is typically returned in a natural language form (sometimes with slight rephrasing) to ensure the result is accurate and easy to understand.[23]

The question answering model can be classified into extractive and generative approaches, which is different based on how they produce the answer.
Extractive QA systems identify the answer within a document by selecting and returning an existing sequence of text. For example, model like BERT can be fine-tuned to precisely detect answer spans in benchmarks such as SQuAD [16].
On the other hand, **Generative QA** systems produce answer in natural language by synthesizing information learned from large training corpora, as seen in large language models like GPT-3, which generate text word based on context. [21]
These approaches leverage the transformer architecture, where the self-attention mechanism enables both a deep understanding of context and the production of coherent and contextually relevant responses.[14] So, extractive QA is ideal when a source-anchored answer is required. While, generative QA is particularly useful for open-ended questions that demand articulated and synthesized responses.

Chapter 6

XML Generation: ETL Automation with AI

This chapter explores the focus of the project carried out at Mediamente Consulting, aiming to automate the creation of parallel jobs in IBM InfoSphere DataStage. This initiative marks a significant advancement in data integration processes which traditionally require extensive manual set up and configuration. By leveraging custom scripts and advanced programming techniques, the project aim to reduce the time and complexity associated with defining, deploying and managing ETL tasks in complex data environments.

The automation scripts are written in Python and utilize XML structures to define job specifications and configurations dynamically. This scripts interacts with Datastage's underlying systems to programmatically set up data connections, specify data transformations and orchestrate job sequences. This approach not only wants to deal with the scalability and reproducibility of data workflow, but also it significantly reduces the potential for human error and the overall time needed for job set up.

The subsequent sections provide a comprehensive overview of the scripting techniques employed, the architecture of the automation solution and a detailed analysis of the algorithms in the scripts, highlighting how these interact with DataStage components to reproduce automation. Additionally the chapter discusses the challenges encountered during implementation and the solution implemented, offering insights in the practical aspects of automating data integration tasks in a corporate setting.

6.1 Component creation with Datastage

The automation of DataStage jobs starts with a fundamental understanding of the tool's operations and its potential for automation, particularly in the context of the company's existing data management framework as outlined in Chapter3.1 . Initially, the first step involved creating a standard job into DataStage to serve as a baseline for automation scripts. This initial step is critical because it estabilishes a template from which all the next automated tasks come from.

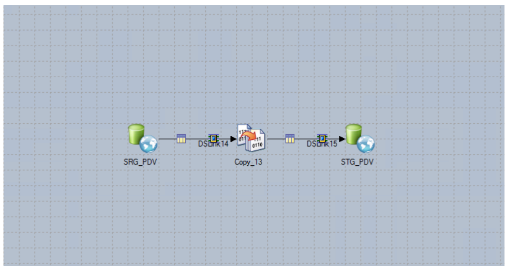


Figure 6.1. STG stage in DataStage

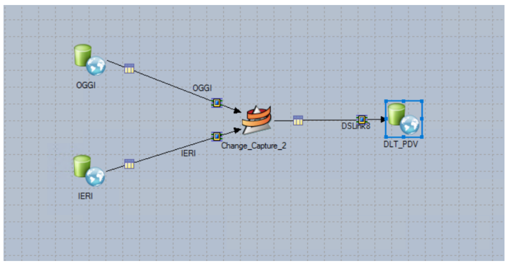


Figure 6.2. DLT stage in DataStage

Once the job is configured and operational, it is exported to an XML format. Exporting the job as XML is pivotal because it provides a comprehensive blueprint of the job's structure, including details about its configuration and the inter dependencies of its components. This XML is crucial as an artifact for understanding the static and dynamic aspects of the job configuration.

The next phase consists in a detailed analysis of the exported XML to identify which parameters are constant across similar jobs and which parameters need to be dynamically adjusted to prepare the job for the specific requests of the user. This analysis is essential for developing scripts capable to modify the XML in a controlled and predictable manner. The parameters typically subject to modification include:

- **Job names and identifiers:** unique identifiers for each job.
- **Data connections:** sources and destinations for data flows which are very dependent on the project or the specific task.
- **Transformation rules:** specific data manipulation tasks within the job

The parametrization of the elements in the automation scripts makes it possible to deploy customized jobs without manual intervention. To do this, placeholders are used in the XML structure, replaced by user-specific or application-specific data. With this approach, it is possible to reduce not only the setup time for new jobs but also enhance the adaptability of the data integration infrastructure to meet evolving business needs.

Furthermore, the automation scripts are designed in order to be reusable and easily adaptable, allowing for quick modifications and updates to the job configurations as new requirements emerge. This flexibility is fundamental to maintain efficiency in a dynamic business environment, where data sources , business rule and data targets frequently change.
In conclusion, the ability to automate DataStage job creation using XML templates, significantly simplifies the process of setting up, modifying and deploying ETL jobs. This method reduces the manual effort required, decreasing the potential for errors and allowing data teams to focus more on strategic tasks rather than repetitive configuration.

6.2 Table structure in excel

This is a necessary step in order to gather information from the client and compose the attributes of the database tables, used at various stage of the project.
This process is needed for a documentation to support the automation process on the DataStage platform, which relies on predefined mapping structures for data manipulation. This automation process includes the mapping in a sequential manner for algorithmic processing.
This excel setup has been customized to align with the specific requirements of DataStage workflows. Adjustments include integration of fields used to identify primary keys of table and in order to manage the expressions necessary for generating new data fields,which are transferred sequentially from a table to another. These fields are for example, `INS_TIME` or `JOBID`. This excel file is a crucial part for the creation of the custom XML files in order to generate new jobs. It contains, in addition to the name of the fields, specifications such as data type, length, scale and precision. Despite it is one of the manual tasks in the workflow, it offers a standardized structure which minimizes the effort required by users in the compilation of it.

6.3 Construction of database

This JSON-based repository, kept in a simple file format for convenience, is created to store user instructions alongside the corresponding function calls required by DataStage to fulfill those instructions. While the JSON file itself is not a vector database, its records will be converted in next steps into vector embeddings to enable semantic searches. Concretely, each record in the file consists of two fields:

1. **request**: a textual description of a task that the user want to perform (for example, from simple request as "Create a job with name JOB" or a more complex like "Create a job for the STG phase called `data_flow` with two ODBC connectors `data_source` and `data_target` and a copy stage called `data transfer`")
2. **response**: a specific function call that represents how DataStage should create or configure the job in question (for example `"create_job_only('JOB')"` or `"create_copy_job_odbc('data_flow', 'data_source', 'data_transfer', 'data_target')"`)

In this way, the organization of the database becomes the cornerstone of the automation pipeline, enabling the system to interpret user queries, retrieving matching records and producing the correct DataStage job creation code with minimal manual effort. Naturally, not all the possible types of stages configuration and connectors that it is possible to create with DataStage, are inserted in the Database, but only those common and necessary for the company's workflows, as described in Chapter 3.1.

6.3.1 Splitting of dataset: train and test

When the splitting of datasets works in a situation like this, it is very important to not fall in situation of overfitting or underfitting. In the preparation of dataset of user requests for DataStage job generation, the entries are first labeled according to the different functions, as said in the previous paragraph. These labels ensure that each request can be associated with the corresponding transformation or connector logic. Despite using a random approach to divide the dataset, the script used to split it attempts a stratified division. In fact, it checks the distribution of labels and tries to preserve their portions in training and test sets. This approach is crucial to maintains a similar variety of function calls in both sets, preventing any single category from becoming underrepresented.
Therefore, after splitting, there is a training subset (`VectDB_train.json`) used to guide the model's learning about which function corresponds or which type of request. Conversely, the test subset (`VectDB_test.json`) is preserved in order to evaluate the model and see how good it generalizes when there are unseen requests. The preservation of the label variety as much as possible in both subsets, helps ensuring that the model perform robustly with the different DataStage job configurations it will encounter.

6.4 Flow implementation of the automation

In this section the overall architecture of the designed automation system is presented. In figure 6.3 it is possible to visualize the main steps to implement the pipeline which is described in the next subsection.

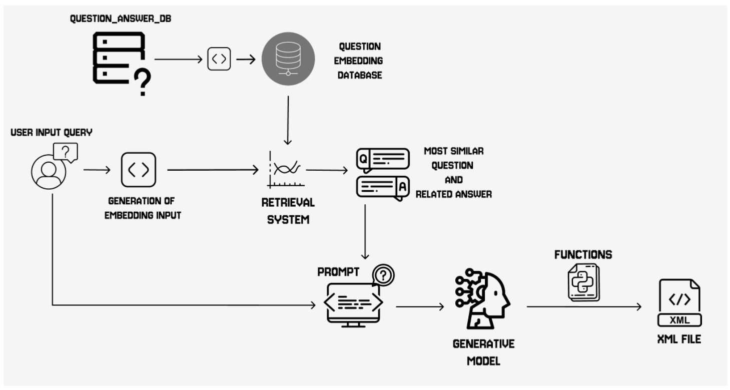

QUESTION_ANSWER_DB
QUESTION EMBEDDING DATABASE
USER INPUT QUERY
GENERATION OF EMBEDDING INPUT
RETRIEVAL SYSTEM
MOST SIMILAR QUESTION AND RELATED ANSWER
PROMPT
GENERATIVE MODEL
FUNCTIONS
XML
XML FILE

Figure 6.3. Pipeline implementation

6.5 Generation of embeddings

In this section it is explained the part of the project regarding the generation of embeddings from the text.
In the context of Natural Language Processing (NLP) mentioned above, an embedding is a numeric representation of a text (phrase,word or document) in a continuous vector space. The idea is to capture semantic of text, similar phrases or similar words might be near in this space. Practically, it allows to facilitate the comparison between text queries, perform similarity searches or feed the vector into machine learning and language analysis algorithms.
The database explained before contains a series or text-based requests: for each of these strings it is calculated a corresponding value (embedding). In this way, results enrich database adding a structure (typically an array of floating-point numbers) which represents its meaning.

In this process there are three main phases :

1. Before generate the embedding, a preprocessing stage is performed and each request undergoes a cleaning pipeline (in order to remove unnecessary punctuation and stopwords), normalization and keyword recognition. This step simplifies the text without loosing important information.

2. Once the tokens are cleaned, it is possible to switch to the embedding generation part, different techniques are used to convert tokens into numbers:
   - **Word2Vec standard:** trained on the requests , each word is associated with a fixed-sized vector. Then, the words in a sentence are combined (for example taking the average) to obtain a single sentence embedding.
   - **Word2Vec with TF-IDF weights:** with respect to the standard Word2Vec, it is given more importance to certain terms. In this case, TF-IDF technique prioritizes words that occur less frequently but have higher significance.
   - **Word2Vec with custom weights:** in this case, as before for TF-IDF, there is an extension of the standard Word2Vec, but there are no weights given in a statistical manner, so they are customized based on domain knowledge.
   - **BERT:** it is a pre-trained transformer model on large amounts of text. It extracts a single embedding for the entire sentence, capturing the complex contexts and relationships between words.
   - **Fast-text standard:** it incorporates subword information, which can be particularly beneficial for handling out-of-vocabulary or morphologically rich words.
   - **Fast-text with TF-IDF weights:** like it happens for Word2Vec it is given more importance to those words which less frequently appear, but they have an higher importance.
   - **Fast-text with custom weights:** in this case, the weights of different words are setting in a custom manner in a file, containing all the different weights.

3. Finally, the resulting vector is stored within the same database (in JSON format), adding an *'embedding'* field to each record.

   This technique is a sort of *feature extraction* where each word or sentence is associated with a numeric representation. Differently from simple 'bag of words' representations , embeddings aim to preserve semantic relationships. Thus, words/sentences with similar meanings generate vectors that are close together,regardless of syntactic form or verb tense. During DB analysis or querying , working with vectors make it possible to use a variety of quantitative techniques (for example distance metrics, similarity) which are far more effective than only evaluate text matching. Therefore, this technique has proven to be useful in the **semantic search**; in fact, once every question is transformed into a vector , it is possible to quickly compare similarity between requests (using cosine similarity). This is very useful since different requests with similar concepts appear close to each other in the vector space. Moreover by computing embeddings just once and saving them in the DB, it is not needed to recompute them for each query: this will greatly improves performances.

6.6 Implementation of retrieval system

The retrieval system implemented consists in the semantic identification of the most similar queries among those in the database created with respect to the new queries sent as input by the user. The system through a embedding representation of queries, can calculate the similarity between the user requests and the request in the database.
It is organized in several phases:

1. **Load or train of the model:** according to the selected technique, the system loads an embedding model that is just trained and saved. If the model is not saved, it trains a new model. Both for **Word2Vec** and **FastText**. It works differently for model based on **BERT**, which is a pre-trained model and generate the embedding without the need of a local train.
2. **Calculating the Query Embedding:** the user query is converted into an embedding vector using the specified method.
3. **Comparison with the queries in the database:** the embedding calculated previously, now are loaded, extracted and converted in a Numpy array to be compared with the new query.

By each new query given as input by the user, for each method, the results are three different queries returned from the database, sorted in decreasing order based on the most similar to the one given as input by the user.

6.6.1 Similarity methods implemented

In order to calculate the similarities between the queries, three different methods are used, the cosine similarity, the euclidean distance and the manhattan distance. The formulas are shown below:

Cosine Similarity: Given two vector x e y , cosine similarity is defined as:

$$\text{sim}(x,y) = \frac{x \cdot y}{\|x\|\|y\|}$$

Euclidean Distance: Euclidean distance between x e y is calculated with the following formula:

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Manhattan Distance: Manhattan distance is given by :

$$d(x,y) = \sum_{i=1}^{n} |x_i - y_i|$$

Among the three metrics examined, cosine similarity is often considered to be the most effective as it measures the angle between two vectors and it is therefore independent of their magnitude. In contrast, Euclidean distance is useful in spaces where all dimensions are equally significant. Its reliability can suffer in high-dimensional contexts due to the "curse of dimensionality", where sparse data and numerous variables can distort the results. Manhattan distance, on the other hand, tends to perform better than Euclidean distance in high-dimensional or sparse settings because it aggregates the absolute differences linearly across dimensions, although it still remains sensitive to the overall scale of the vectors.

6.7 XML generation

For the generation of the XML files, a script is implemented, the process has been structured in a modular way to facilitate the management, reuse and maintenance of the code.

In the code, placeholders are fundamental elements for the dynamic generation of XML files. They are values replaced when the code is executed, ensuring that the final XML file contains the correct and complete information.

6.7.1 Description of function and large language model: Gemini

One of the main step of the project pipeline is the use of generative model, in order to make the process really simple and useful for the user. Through it, it is possible with a simple request to Gemini to receive as response the function with all parameters processed and returned at the end an XML files that is ready to be imported in DataStage to create the job the user need.

Gemini is a Large Language Model (LLM) developed by GoogleAI, trained on a massive dataset of text and code. It can generate text and translate languages, write different kinds of creative contents and answer your questions in an informative way.

There are several different gemini models with different strengths and capabilities. The gemini model tested are:

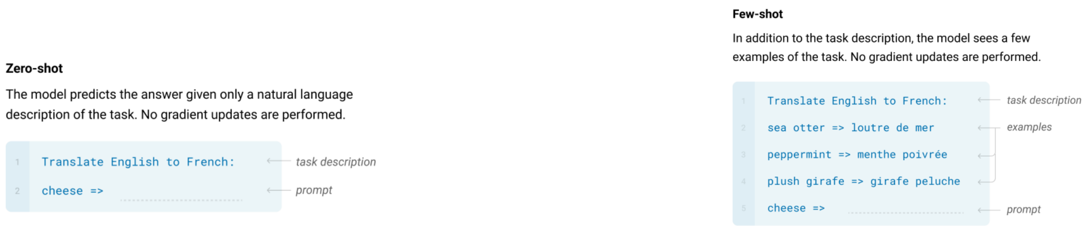
- **Gemini 2.0 Pro Exp**: it is the largest and most capable Gemini model. It is well suited for a wide range of tasks, including code generation, text summarization and question answering.
- **gemini-2.0-flash**: this is a faster and more efficient version of Gemini Pro. It is ideal for tasks that require real-time interaction, such as chatbots and conversation AI.
- **Gemini 2.0 Flash Lite 001**: this model is a highly optimized version of Gemini, designed to be light and fast. "Lite" indicates that it is a scaled-down version, designed for devices with limited resources or for applications that require quick responses. It's ideal for tasks where speed and efficiency are prioritized, although it may have slightly lower capabilities than larger models
- **Gemini 1.5 Flash 8b Latest**: this model is oriented to be fast and the dimension of this model is of 8 billion of parameters. The parameters measure how good is a model to learn complex patterns: thr more parameters , the better the models.

6.7.2 Different prompt techniques implemented

After the identification of the most similar question to the one posed by the user along with its corresponding answer, a critical step in the automation process is reached: prompt creation. Therefore, understanding and mastering different prompt techniques is vital to optimize interactions with the generative model. Below it is possible to analyze the five prompt technique used:

1. **Zero-Shot**: it consists of asking the model a task or question in natural language, without providing any examples, the prompt contains only the description of the activity and the model has to produce the answer by drawing exclusively on the knowledge learned during the pre training. Furthermore on very complex or specialized tasks, the model may not possess sufficient knowledge and in these cases, zero-shot results in incorrect or superficial responses.[15]

2. **Few-Shot**: this technique consists in providing in the prompt some examples as input with the corresponding answers before asking the new question. The model, observing these demonstrations, conditions its own generation based on them and produces an output that follows the pattern learned from the few examples provided. Compared to zero-shot , it is generally more effective precisely because of the provided examples to the model. Despite being powerful, it presents some critical issues; in fact, accuracy can vary greatly depending on which examples are presented and in what order. Indeed, it has been observed that rearranging the examples, using slightly different formulations, or even including unrelated examples, can significantly affect the outcome. Furthermore, the few-shot model does not learn permanently: if it is necessary to answer to many queries of the same type, include example in every prompt can become inefficient. [29]


Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
Translate English to French:
cheese =>
```

Few-shot

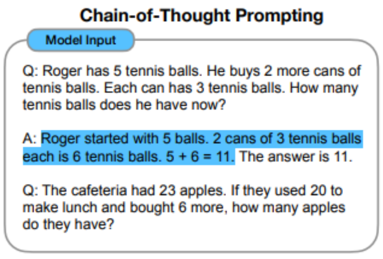
In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
Translate English to French:
sea otter => loutre de mer
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese =>
```

Figure 6.4. comparison between zero-shot and few-shot example prompt [15]

3. **Chain-of-Thought (CoT)**: with this technique the model is trained to generate intermediate reasoning steps before providing the final answer. Instead of just asking the model for the answer, the prompt is formulated in such a way that the model "thinks aloud", outlining a logical sequence of deductions.
   The purpose of CoT prompting is to assist the model in breaking down complex problems into simpler parts,making the solution process explicit before providing

the final output.
Some limitations of this prompt technique are that it requires sufficiently large models: in fact, below a certain threshold of parameters, the model is unable to produce useful reasoning chains. Another issue may be that if the model makes a logical error in one of the intermediate steps, this can compromise the final response. Another aspect is the computational cost: generating detailed explanations means using many more tokens for each query, which can reduce speed and increase the usage costs of the model.[35]



Chain-of-Thought Prompting

Model Input

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?

Figure 6.5. example of CoT prompt technique [35]

4. **Iterative Refinement**: it is a technique where a model incrementally improves its outputs through sucessive cycles of self-assessment and modification. This process allows the model to enhance the quality and accuracy of its responses by continuously refining its initial outputs. For instance, the ISR-LLM framework employs iterative self-refinement to boost the planning capabilities of large language models (LLMs) in complex, long-term sequential tasks. By translating inputs from natural language into a structured planning language and iteratively refining the generated plans, ISR-LLM (iterative Self Refined Large Language Model) achieves higher success rates compared to traditional LLM-based planners. Similarly, the COrAL(Context-Wise Order-Agnostic Language Modeling) approach integrates iterative refinement directly into the LLM architecture, enabling the model to internally improve its outputs during the generation process. This method captures diverse dependencies without incurring the high inference costs associated with sequential generation, leading to improved performance and efficiency in reasoning tasks. [36][33]

5. **Task Decomposition**: it involves transforming complex tasks into smaller, manageable sub-tasks that can be addressed individually to achieve the overall objective. This strategy leverages the strengths of LLMs in managing discrete components of a problem, facilitating more effective problem-solving. This dynamic adjustment leads to significant improvements in success rates across various interactive decision-making tasks. Furthermore, the Task Navigator framework extends multimodal LLMs to tackle complex tasks by breaking them down into vision-related sub-questions. This method integrates a refinement process to ensure the feasibility and responsiveness of the sub-questions, enabling LLMs to provide accurate responses for intricate tasks. [37] [34]

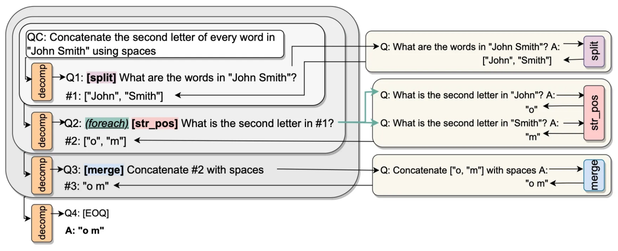

QC: Concatenate the second letter of every word in "John Smith" using spaces
decomp
Q1: [split] What are the words in "John Smith"?
#1: ["John", "Smith"]
decomp
Q2: (foreach) [str_pos] What is the second letter in #1?
#2: ["o", "m"]
decomp
Q3: [merge] Concatenate #2 with spaces
#3: "o m"
decomp
Q4: [EOQ]
A: "o m"
Q: What are the words in "John Smith"? A: ["John", "Smith"]
split
Q: What is the second letter in "John"? A: "o"
Q: What is the second letter in "Smith"? A: "m"
str_pos
Q: Concatenate ["o, "m"] with spaces A: "o m"
merge

Figure 6.6. example of Task decomposition prompt flow [34]

These are only a subset of all types of prompt techniques available, but they clearly illustrate how even subtle variations in prompt design can significantly influence a model's output. By understanding, tailoring and even combining these strategies, performance can be optimized, response consistency enhanced and the full potential of large language models can be unlocked for a variety of complex tasks.

Next page contains a snippet which encapsulates all the entire process of the prompt engineering developed and the creation of the XML file through the generative model in which is possible to distinguish 5 steps:

- *Step 1* **Retrieving similar examples:** is used the *retrieve_all_similarities* function in order to search in the database a pair of examples (question-answer) semantically close to the user's query.
- *Step 2* **Few-Shot prompt construction:** the retrieved examples are included into a Few-Shot prompt in order to provide the context for the prompt.
- *Step 3* **Gemini API call:** the prompt is sent to gemini including some parameters such as *temperature*, *max_output_tokens* and *max_retries* in order to generate a response including the python code.
- *Step 4* **Execution and saving of the generated code:** the python code is executed in order to produce the XML file to import in DataStage and it is saved to disk in a defined location, completing the process.

Algorithm 1 XML Generation Process using Gemini API with Few-Shot Prompting

Listing 6.1. High-level process for generating and saving an XML file

```
def generate_xml_from_request(user_request):

## –– STEP 1 –– ##
––– Retrieve similar questions from question embedding database –––
  results = retrieve_all_similarities(use_request, candidate_db_filepath, top_k = 2)

     ––– Example of similar retrieved question –––

    similar_question1 = Design an Oracle integration by generating two connectors,
        PRIMARY_ACCOUNT and SECONDARY_ACCOUNT, for the job financial_sync with a Copy
        stage named update_financial for the STG phase
    similar_answer1= create_copy_job_oracle('financial_sync', 'PRIMARY_ACCOUNT', '
        update_financial', 'SECONDARY_ACCOUNT')

    similar_question2 =
    Create a job for the STG phase called data_flow with two ODBC connectors data_source
        and data_target and a Copy stage called data_transfer
    similar_answer2 =
    create_copy_job_odbc('data_flow', 'data_source', 'data_transfer', 'data_target')

## –– STEP 2 –– ##
––– Construct the few-shot prompt using the similar question retrieved –––
    prompt = f"""
    Example 1:
    Q: {similar_question1}
    A: {similar_answer1}

    Example 2:
    Q: {similar_question2}
    A: {similar_answer2}

    Given this two example above and this new question: {user_request}
    write the answer associated to the new question.

## –– STEP 3 –– ##
    ––– API Call to Gemini –––
    response = call_gemini_api(
        prompt,
        temperature=0.5,
        max_output_tokens=1000,
        max_retries=5
    )
    code_response = clean_code_response(response.text)

## –– STEP 4 –– #
     ––– Execute the Generated Code to Produce XML –––
      ––– Save the Generated XML File –––
    complete_code = "result = " + code_response.strip()
    local_vars = {}
    exec(complete_code, globals(), local_vars)
    xml_content = local_vars.get("result")

    output_filepath = "C:/path/to/output.xml"
    save_xml_to_file(xml_content, output_filepath)
    return xml_content

 ––– Example usage: –––
user_request = (I need an STG phase called staging,
                two Oracle connectors (source and target),
                and a copy stage named transfer with ODBC connectors)
xml_generated = generate_xml_from_request(user_request)
```

6.8 Results

This section presents the results of the various techniques and models used and experimented within the pipeline. Embedding techniques are the backbone of the retrieval of similar queries in the database, their evaluation of the text is the beginning of the analysis. The embeddings serve as the foundation for the generative model, enabling it to produce more context-sensitive and relevant results. The evaluation of the performance of the generative model, which examines how well it uses the retrieved similar queries to generate accurate and meaningful results, comes next in the chapter. The purpose of the analysis is to compare various configurations of the parameters and how each influences the overall performance, with the ultimate purpose of identifying the best setup for the pipeline.

6.8.1 Evaluation of the different text embedding models

To calculate the similarities between the queries, there are three different methods usable: cosine similarity, Euclidean distance and Manhattan distance. The formulas are summarized in Table 6.1.

Metric	**Formula**
Cosine Similarity	$\text{sim}(x, y) = \frac{x \cdot y}{\|x\|\|y\|}$
Euclidean Distance	$d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$
Manhattan Distance	$d(x, y) = \sum_{i=1}^{n} \lvert x_i - y_i \rvert$

Table 6.1. Summary of Similarity Metrics Used for Retrieval

Among these similarity measures, the `cosine similarity` is particularly effective, as it accurately captures the semantic similarity between text embeddings without being influenced by vector magnitudes. Although `Euclidean distance` and `Manhattan distances` provide valuable alternatives based on geometric proximity. They tend to be more sensitive to the absolute magnitudes of embedding vectors, potentially diminishing their effectiveness in high-dimensional or sparse embedding spaces.

While similarity measures are very imporatnt in the ranking phase to select the most relevant candidate responses, the last evaluation of retrieval effectiveness is based on metrics such as `Precision@K`, `Recall@K` and `Top-1 Accuracy`, which quantify the quality of retrieved results in a practical manner. These metrics measure the effectiveness of the model in including correct responses within the top ranked positions, providing concrete indicators of model performance.

- **Precision@K** quantifies the proportion of relevant results among the top K retrieved candidates:

$$\text{Precision@K} = \frac{\text{Number of relevant responses in top } K}{K}$$

  This metric highlights the effectiveness of the model in ranking the correct or relevant responses at the top of the list. A higher value indicates better performance.

- **Recall@K** is a binary measure defined as:

$$\text{Recall@K} = \begin{cases} 1, & \text{if the expected response is among the top } K \text{ retrieved responses,} \\ 0, & \text{otherwise.} \end{cases}$$

  By averaging this binary indicator over all queries, it is obtained an overall measure of how often the correct response is retrieved within the top K candidates.

- **Top-1 Accuracy** is defined as:

$$\text{Top-1 Accuracy} = \frac{\text{Number of queries for which the top ranked response is correct}}{\text{Total number of queries}}$$

  This metric provides insight into the model's ability to place the correct response in the very first position of its ranking.

The performance of each method is evaluated on a test set and the following table summarizes the final results.

Method	**Precision@K**	**Recall@K**	**Top-1 Accuracy**
Word2Vec Standard	0.61	1.00	0.75
Word2Vec TF-IDF	0.54	0.96	0.63
Word2Vec Custom	**0.82**	**1.00**	**0.88**
BERT	0.49	0.83	0.63
FastText Standard	0.61	1.00	0.75
FastText TF-IDF	0.67	1.00	0.71
FastText Custom	0.78	1.00	0.83

Table 6.2. Final Comparison of Text Embedding Methods

The experimental results demonstrate that the **Word2Vec Custom** model outperforms the other methods across all three key metrics. In particular, it achieves a Precision@K of 0.8194, a Recall@K of 1.0000 and a Top-1 Accuracy of 0.8750. These results indicate that this model is highly effective in retrieving relevant queries and placing the correct response at the top of the list.

One plausible explanation for the superior performance of the Word2Vec Custom method is its tailored weighting strategy. Unlike the standard Word2Vec model, which assigns equal importance to all words, the custom variant adjusts the weights of the individual word embeddings. This adjustment allows the model to emphasize more informative or discriminative words that are crucial for capturing the semantic content of the text. As a consequence, the resulting embeddings are more representative of the true meaning of the queries, leading to better similarity measurements.

When compared to the Word2Vec TF-IDF approach, the custom method still shows a marked improvement. Although TF-IDF incorporates an inverse document frequency component to highlight rarer terms, it may not fully capture the domain-specific importance of certain words. The custom weighting scheme, on the other hand, can be designed to reflect nuances specific to the application domain, which likely contributes to its enhanced performance.

Moreover, while models based on BERT and FastText also provide strong contextual embeddings, they do not match the retrieval performance of the Word2Vec Custom model. BERT, despite its deep contextual understanding, is primarily designed for general language understanding tasks and might not be optimized for the precise similarity matching required in this retrieval scenario. Similarly, while the FastText Custom model also benefits from a custom weighting strategy, its overall performance still lags slightly behind the Word2Vec Custom method, further emphasizing the effectiveness of the latter's approach.

Thus, in conclusion the analysis confirms that employing a custom weighting strategy within the Word2Vec framework yields significant improvements in retrieval tasks. The Word2Vec Custom model consistently outperforms other variants and alternative embedding approaches in terms of Precision@K, Recall@K and Top-1 Accuracy. This study highlights the potential of tailored embedding strategies for enhancing the performance of text retrieval systems. Future research may extend these findings by exploring further refinements in weighting mechanisms and integrating additional context-aware features to push the performance envelope even higher.

6.8.2 Evaluation of the different prompting techniques

The results presented below in figures 6.11 and 6.14 offer meaningful insights into the relationship between different prompting techniques, their combinations and the performance of different Gemini models. Specifically, the metrics analyzed are the average accuracy percentage in correctly responding to a set of requests (which included both clearly and ambiguous prompts) and the average response times associated with each prompting technique and model.

Firstly, it is immediately evident that **Few-Shot + Iterative Refinement** achieve the highest accuracy with a success rate of more than `90%` across both gemini-2.0-flash and gemini-2.0-flash-lite-001. The strong performance observed with this combination can be attributed to two primary factors. This represents a significant improvement compare to the **Few-Shot** technique alone, which has a success rate of `85%`. This increment underscores the added value of iterative refinement when integrated into a Few-Shot context. This improvement likely results from iterative refinement's ability to systematically refine model responses through a cyclical feedback mechanism. In fact, the iterative refinement allows model to progressively approach an optimal solution by reevaluating previous outputs and incrementally improving accuracy based on intermediate feedback. Thus, the high success rates obtained are aligned with theoretical expectations about the effectiveness of combining few-shot exemplars with iterative validation processes. Interestingly, we observed that Few-Shot alone slightly outperforms Few-Shot combined with Task Decomposition. This can occur because task decomposition, although generally beneficial for complex tasks, may introduce unnecessary complexity in simpler scenarios, leading the model to split its focus and slightly decrease performance.

Similarly, the combination **Chain-of-Thought(CoT) with Iterative Refinement** maintains strong performance, achieving approximately `88%` success for Gemini 2.0 model and little bit more with Gemini 2.0 light model. The CoT technique explicitly instructs the model to reason step-by-step,breaking down complex requests and thus improving comprehension and subsequent answer quality. When combined with iterative refinement this technique can benefit from incremental improvements as it prompts the model to repeatedly evaluate its reasoning steps, ensuring consistency and robustness in final results. However, CoT alone, without iterative refinement, demonstrates less stable performance, ranging from `65.22% to 82.61%` depending on the model and it even drops to `47.83%` with the 1.5 model. It also results in notably longer average response times (`1.63 to 2.55 seconds`). This suggests that, while explicit reasoning can lead to improved accuracy, it may impose a computational overhead due to the increased complexity of intermediate steps.

Comparing the simpler techniques, a clear under performance emerges with the **Zero-Shot** configurations. It systematically resulted in a 0% success rate, demonstrating complete inadequacy for accurately resolving even moderately ambiguous queries. This finding aligns with the idea that Zero-Shot prompting has no effectiveness in scenario with ambiguity or under-specified requests when used alone, without providing any examples. This technique is also further disadvantaged by having the longest response times recorded (up to `3.29 seconds` for the gemini-2.0-flash-lite-001 model), reflecting the potential difficulty the model experiences in processing under-constrained instructions.However, introducing techniques such as Task Decomposition or Iterative Refinement provides a noticeable improvement, highlighting the importance of structured prompting even without initial examples. Nevertheless, the older Gemini 1.5 model shows limited improvement, likely due to its lower capability in managing ambiguity without explicit guidance or context.

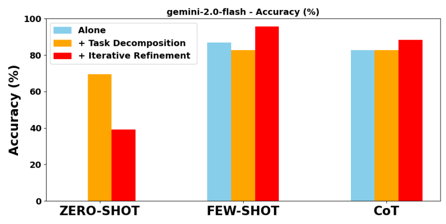


Figure 6.7. Accuracy for gemini-2.0-flash

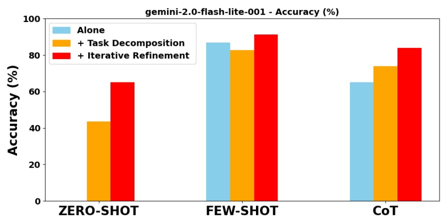


Figure 6.8. Accuracy for gemini-2.0-flash-lite-001

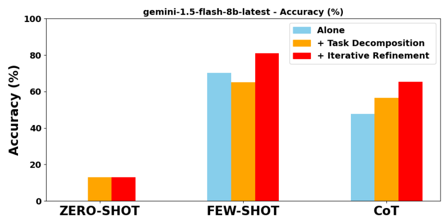


Figure 6.9. Accuracy for gemini-1.5-flash-8b-latest

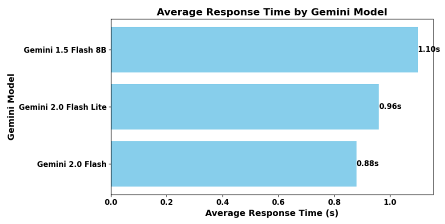


Figure 6.10. Average response time of different models

Figure 6.11. Accuracy for different type of gemini models and average response time

Instead, considering the two prompting strategies **Task Decomposition** and **Iterative Refinement** alone, it is noticeable that individually they produce moderately effective results, especially iterative refinement, which reached approximately 60% success with the lighter Gemini models. Interestingly, Task Decomposition alone shows varied performance depending on the model, in fact while reaching a good accuracy with the model gemini-2.0-flash-lite-001 and gemini-2.0-flash, it experiences significant drops with gemini-1.5-flash-8b-latest (about 40/50% less of accuracy). This highlights how the capability of certain models to handle decomposed sub-tasks can differ significantly, confirming literature findings that decomposition can be effective if properly supported by powerful models capable of context retention.

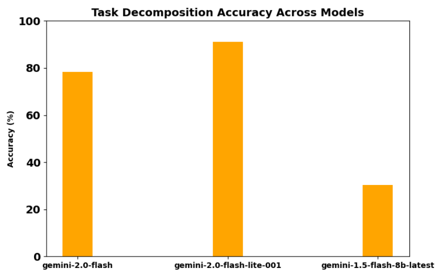


Figure 6.12. Accuracy for Task Decomposition

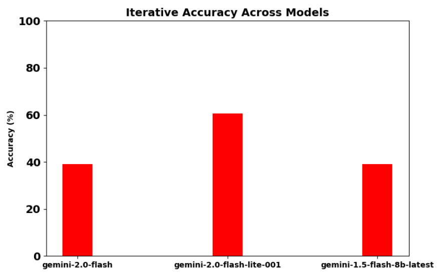


Figure 6.13. Accuracy for Iterative Refinement

Figure 6.14. Comparison between Iterative Refinement and Task Decomposition

The choice of the Gemini model has a very important influence on both the accuracy and response time. For instance, models like gemini-2.0-flash and gemini-2.0-flash-lite-001 consistently delivery good accuracy and remarkably fast response times (typically **under 1 second**), whereas the older `gemini-1.5-flash-8b-latest` frequently produces lower accuracy and also an higher average latency . The variations in accuracy and speed across models likely derive from differences in parameter counts, training data and underlying architecture optimizations.
The Gemini 1.5 model consistently shows lower performance even when employing advanced prompting techniques. This limited effectiveness is primarily due to its lower number of parameters and less advanced training methods, leading to weaker semantic understanding and reduced context retention compared to newer Gemini 2.0 models.

Furthermore, some combinations of prompt techniques provide minimal different output. Techniques like Zero-Shot combined with Iterative Refinement and task decomposition consistently underperformed, suggesting limited benefit from refinement steps in absence of context. Similarly, Few-Shot used with Task Decomposition provides lower accuracy improvements compared to used with Iterative Refinement, further reinforcing the crucial role iterative feedback plays in elevating outcomes, rather than mere task breakdown alone.

Finally, the most advanced model proposed by gemini, the `gemini-2.0-pro-exp`, does not be evaluated with the same extension and with all the different queries used by the other models. However, a clear trend has emerges from the few tests conducted : although the responses required more time, their quality is decidedly superior. It is therefore reasonable to hypothesize that, if additional resources had been available for an extensive test, the "pro" model would have consistently ranked at the top for accuracy and reliability, especially in the more complex cases.

In conclusion these results align with theoretical expectations from recent studies, reinforcing the importance of context-rich prompting methods and iterative correction processes. They highlight the substantial advantage provided by combining Few-Shot prompting with Iterative Refinement, while clearly illustrating the limits of simpler techniques (Zero-shot) when dealing with ambiguous user requests. Moreover, these findings underscore the critical role played by the choice of LLM in balancing accuracy and computational efficiency, reinforcing the importance of selecting both the appropriate prompting strategy and an optimized language model to achieve the desired performance outcomes. In the table 6.3 it is possible to see the aggregated results by prompt technique, in order to see , independent from the model, which is the best prompt technique to use, both in term of accuracy with respect to the given answer and the time, that is the time the model need to generate the response.

Prompting Technique	**Average Response Time (s)**	**Average Accuracy (%)**
Zero-Shot	2.25	0.0
Zero-Shot + Task Decomposition	0.64	42.01
Zero-Shot + Iterative Refinement	0.91	39.13
Few-Shot	0.62	81.37
Few-Shot + Task Decomposition	0.59	76.81
Few-Shot + Iterative Refinement	**0.63**	**89.27**
Chain-of-thought	2.06	65.22
Chain-of-Thought + task decomposition	0.66	71.04
Chain-of-Thought + Iterative Refinement	0.65	79.24
Iterative Refinement	0.60	46.26
Task Decomposition	0.62	66,56

Table 6.3. Aggregated Results by Prompt Technique

Chapter 7

Conclusions and future works

This thesis presents a comprehensive approach towards automating ETL (Extract, Transform, Load) process using advanced artificial intelligence techniques, focusing specifically on the generation of XML job configurations within the IBM DataStage environment. The main goal is to significantly reduce manual configuration effort, minimize potential human errors and improve the overall flexibility and responsiveness of ETL processes in adapting to evolving data management requirements.
The developed solution explores sophisticated text embedding methods and advanced prompting techniques, employing state-of-the-art generative models such as Gemini to translate natural language user requests into structured, executable XML templates. The research demonstrated that integrating customized embedding models, particularly the Word2Vec Custom approach, with Few-Shot prompting combined with Iterative Refinement provides the highest accuracy and optimal computational performance. This combination allows for the accurate interpretation of complex user requirements, ensuring that generated DataStage jobs closely match user specifications and operational needs.
The experimentation demonstrates a significant advantage in operational efficiency, reliability and adaptability, confirming the effectiveness of AI-driven automation in ETL job creation. This approach not only streamlines workflows but also provides organizations with a powerful tool for rapidly responding to new data integration scenarios, a crucial capability in the rapidly evolving landscape of business intelligence and analytics.

Regarding the future works, this thesis opens several way for future research and development, with significant potential for further improvements and extensions of the current automated ETL system. Some future developments that could be done are described below:

Automatic Excel Sheet Generation
A natural future progression of this research would be to fully automate the creation of the Excel sheets which is now compiled manually. By extending the Generative AI techniques employed in this thesis, future development could automatically produce and update Excel sheets based on initial user inputs. This would further minimizes manual intervention

and improves the reliability of initial data entry and configuration tasks, reducing human-induced errors and enhancing operational accuracy even if the excel is simple to compile for an user also without automation.

Extension of database and model exploration
Another things that could be improved in the future is the expansion of the existing database to incorporate a broader range of ETL configurations beyond those currently used by the company. By enriching the training dataset with a wider variety of configurations, the system could generalize better to different enterprise contexts, making it applicable to broader set of business scenarios. This would involve the continuous integration of new user requests and examples into the system, further refining embedding models and enhancing overall flexibility.

User interface and usability improvements
Another future direction could be the development of a more user-friendly interface or chatbot enabling end users to interact with the ETL automation system more intuitively. By providing an easy-to-use, conversational interface leveraging NLP, users without technical knowledge could directly define complex data integration processes, further democratizing the ETL automation capability across business units.

Fine-Tuning of generative models for enhanced performance
Lastly, an important future development could involve performing fine-tuning of the generative models employed in the ETL automation pipeline. The models utilized in this thesis are used without specific tuning on task-specific datasets. Thus, fine-tuning represents a promising method to enhance their precision and contextual accuracy. By training the generative models on a domain-specific corpus consisting of real user requests and associated accurate XML outputs, the system could learn domain-specific terminology, linguistic patterns and technical details more effectively. This would potentially improve the accuracy and reliability of the generated ETL pipeline scripts. Fine-tuning could also reduce response times and enhance the model's ability to handle ambiguities and highly technical instructions, making the overall automation system even more robust and tailored to the precise needs of the data integration scenarios.

Each of these suggested developments have the potential to significantly enhance the efficiency, flexibility and usability of ETL processes, ultimately empowering organizations to fully leverage their data resources and improve decision-making capabilities in an increasingly data-driven business landscape.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] Designing data marts for data warehouses. *ACM Transactions on Software Engineering and Methodology.*

[2] Etl pipeline guide. URL `https://airbyte.com/data-engineering-resources/etl-pipeline`.

[3] Embeddings and llm. URL `https://datasciencedojo.com/blog/embeddings-and-llm/`.

[4] Etl: significato, funzionamento e vantaggi. URL `https://www.dedatech.com/blog/etl-significato-funzionamento-e-vantaggi/`.

[5] Fasttext - working and implementation. URL `https://www.geeksforgeeks.org/fasttext-working-and-implementation/`.

[6] Word embeddings in nlp. URL `https://www.geeksforgeeks.org/word-embeddings-in-nlp/`.

[7] Ibm datastage documentation. URL `https://www.ibm.com/docs/en/ws-and-kc?topic=data-datastage`.

[8] Real-time in-memory oltp and analytics with apache ignite on AWS. URL `https://aws.amazon.com/blogs/big-data/real-time-in-memory-oltp-and-analytics-with-apache-ignite-on-aws/`.

[9] Database olap ed oltp - cosa sono ed a cosa servono. URL `https://managedserver.it/database-olap-ed-oltp-cosa-sono-ed-a-cosa-servono`.

[10] Star schema - data glossary, . URL `https://www.starburst.io/data-glossary/star-schema/`.

[11] Cos'è il modello n-gram? una panoramica completa, . URL `https://it.statisticseasily.com/glossario/cos%27%C3%A8-il-modello-n-gram-una-panoramica-completa/`.

[12] 11 word2vec approaches - word embedding in nlp. URL `https://ayselaydin.medium.com/11-word2vec-approaches-word-embedding-in-nlp-538478c14b37`.

[13] Conceptual model for star schema data warehouse. *International Research Journal of Modernization in Engineering Technology and Science*, 2023. ISSN 2582-5208.

[14] Adam Jatowta Abdelrahman Abdallah, Bhawna Piryani. Exploring the state of the art in legal qa systems. *arXiv preprint arXiv:2304.06623*, 2023. Accessed via `https://arxiv.org/pdf/2304.06623`.

[15] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. URL `https://arxiv.org/abs/2005.14165`.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019. URL `https://arxiv.org/abs/1810.04805`.

[17] Benyamin Ghojogh and A. Ghodsi. Recurrent neural networks and long short-term memory networks: Tutorial and survey. *ArXiv*, abs/2304.11461, 2023. doi: 10.48550/arXiv.2304.11461.

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. ISBN 978-0262035613. URL `https://www.deeplearningbook.org/`.

[19] Jack Grieve, Sara Bartl, Matteo Fuoli, Jason Grafmiller, Weihang Huang, Alejandro Jawerbaum, Akira Murakami, Marcus Perlman, Dana Roemling, and Bodo Winter. The sociolinguistic foundations of language modeling.

[20] Zhen Huang, Haiping Dong, et al. A survey on attention mechanisms in deep learning. *ACM Computing Surveys*, 2020. doi: 10.1145/3439721. URL `https://dl.acm.org/doi/10.1145/3439721`.

[21] IBM. Question answering. `https://www.ibm.com/think/topics/question-answering`. Accessed: 2025-02-26.

[22] IBM. How artificial intelligence is enhancing olap for better business insights, 2023. URL `https://www.ibm.com/think/topics/ai-for-olap`. Accessed: 2025-02-17.

[23] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, 2009. URL `https://web.stanford.edu/~jurafsky/slp3/old_oct19/25.pdf`.

[24] Ralph Kimball and Margy Ross. *The Data Warehouse Toolkit*. Wiley, 2016.

[25] Michael Lock. Angling for insight in today's data lake. October 2017. Senior Vice President, Analytics and Business Intelligence.

[26] Tim Lu. Star schema vs. snowflake schema. URL `https://www.datacamp.com/blog/star-schema-vs-snowflake-schema`.

[27] Aditi Prakash. Star schema vs. snowflake schema: What to choose? URL `https://airbyte.com/data-engineering-resources/star-schema-vs-snowflake-schema`.

[28] Guru Prasad Selvarajan. Integrating machine learning algorithms with olap systems for enhanced predictive analytics. *World Journal of Advanced Research and Reviews*, pages 62–71, 2019. URL `https://wjarr.com/sites/default/files/WJARR-2019-0064.pdf`.

[29] Mikel Artetxe Mike Lewis Hannaneh Hajishirzi Luke Zettlemoyer Sewon Min, Xinxi Lyu Ari Holtzman. Rethinking the role of demonstrations:what makes in-context learning work? *arXiv preprint arXiv:2202.12837*, 2022.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017. URL `https://arxiv.org/abs/1706.03762`.

[31] Sewon Min† Patrick Lewis Ledell Wu Sergey Edunov Danqi Chen Wen-tau Yih] [Vladimir Karpukhin, Barlas Oguz. [dense passage retrieval for open-domain question answering]. *arXiv preprint arXiv:2004.04906*, 2020. Accessed via `https://arxiv.org/pdf/2004.04906`.

[32] Zichong Wang, Zhibo Chu, Thang Viet Doan, Shiwen Ni, Min Yang, and Wenbin Zhang. History, development, and principles of large language models - an introductory survey.

[33] Yuxi Xie, Anirudh Goyal, Xiaobao Wu, Xunjian Yin, Xiao Xu, Min-Yen Kan, Liangming Pan, and William Yang Wang. Coral: Order-agnostic language modeling for efficient iterative refinement. *arXiv preprint arXiv:2410.09675*, 2024.

[34] Zhiting Yao, Maarten Bosma, Yue Zhao, Xuezhi Wang, Ed Chi, Quoc Le, and Denny Zhou. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.02406*, 2022.

[35] Jason Wei Xuezhi Wang Dale Schuurmans Maarten Bosma Brian Ichter Fei Xia Ed H. Chi Quoc V. Le Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 10 Jan 2023.

[36] Zhehua Zhou et al. Isr-llm: Iterative self-refined large language model for long-horizon sequential task planning. *arXiv preprint arXiv:2308.13724*, 2023.

[37] Yuting Zhuang et al. Task navigator: Decomposing complex tasks for multimodal large language models. *arXiv preprint arXiv:2311.05772*, 2023.