School of Information Technology and Engineering at the ADA University

School of Engineering and Applied Science at the George Washington University

COMPUTING INFRASTRUCTURE AND DATA PIPELINE FOR ENTERPRISE-SCALE DATA PREPARATION

A Thesis
Presented to the Graduate Program of Computer Science and Data Analytics
of the School of Information Technology and Engineering
ADA University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Computer Science and Data Analytics
ADA University

By
Sadig Akhund

April, 2023

THESIS ACCEPTANCE

This Thesis by: Sadig Akhund
Entitled: *Computing Infrastructure and Data Pipeline for Enterprise-scale Data Preparation*

has been approved as meeting the requirement for the Degree of Master of Science in Computer Science and Data Analytics of the School of Information Technology and Engineering, ADA University.

Approved:

(Adviser)	(Date)
(Program Director)	(Date)
(Dean)	(Date)

ACADEMIC INTEGRITY STATEMENT

"I affirm that this is my own work, I attributed where I used the work of others, I did not facilitate academic dishonesty for myself or others, and I used only authorized resources for my Thesis, per the ADA University Academic Integrity requirements. If I failed to comply with this statement, I understand consequences will follow my actions. Consequences may range from failing the course to expulsion from the program/university and may include a transcript notation."

Sadig Akhund		4/25/2023
(Full Name)	(Signature)	(Date: DD.MM.YY)

ABSTRACT

In today's data-driven landscape, enterprises face significant challenges in managing and processing massive amounts of data for meaningful insights and informed decision-making. Data preparation, a critical process that converts raw data into a usable format, plays a pivotal role in the data pipeline and significantly impacts downstream data analysis and modeling. However, traditional data preparation methods may struggle to keep up with the increasing volumes and complexity of data, leading to scalability issues, inefficiencies, delays, and suboptimal performance in the data pipeline. This thesis presents a comprehensive scalability optimization study that analyzes and optimizes the data preparation process in enterprise-grade data pipelines. The study begins by analyzing common components of data pipelines and identifying limitations and bottlenecks that hinder scalability. It thoroughly examines existing data preparation methods, tools, and technologies, as well as cutting-edge tools and methodologies such as Apache Nifi, Apache Atlas, and Apache Spark for addressing scalability challenges. The research draws insights from literature, industry practices, and state-of-the-art technologies to propose practical strategies and recommendations for designing a scalable data strategy in an enterprise setting. The study provides actionable insights and recommendations to enhance the performance of data pipelines in enterprise-grade data environments. The paper concludes with a summary of key findings, limitations, and future research directions, emphasizing the need for a well-designed data preparation pipeline that incorporates scalable data ingestion, efficient data transformation, and intelligent data storage strategies to ensure reliable and efficient data processing in enterprises dealing with large volumes of data.

TABLE OF CONTENTS

Chapters

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Explanation
API	Application Programming Interface
BI	Business Intelligence
CWM	Common Warehouse Model
DSN	Data Source Name
DSS	Decision-Support Systems
DVC	Data Version Control
EDW	Enterprise Data Warehouses
ELT	Extract, Load, Transform
ETL	Extract, Transform, Load
ML	Machine Learning
MME	Managed Metadata Environment
OIM	Open Information Model
OLAP	On-Line Analytical Processing
OLTP	Online Transaction Processing
ORC	Optimized Row Columnar
RDBMS	Relational Database Management Systems
SKP	Shared Knowledge Plane

Computing Infrastructure and Data Pipeline for Enterprise-scale Data Preparation

A Scalability Optimization Study

Sadig Akhund

The ADA University, Azerbaijan

The George Washington University, USA

sadigaxund@hotmail.com

In today's data-driven landscape, enterprises face significant challenges in managing and processing massive amounts of data for meaningful insights and informed decision-making. Data preparation, a critical process that converts raw data into a usable format, plays a pivotal role in the data pipeline and significantly impacts downstream data analysis and modeling. However, traditional data preparation methods may struggle to keep up with data's increasing volumes and complexity, leading to scalability issues, inefficiencies, delays, and suboptimal performance in the data pipeline. This thesis presents a comprehensive scalability optimization study that analyzes and optimizes the data preparation process in enterprise-grade data pipelines. The study begins by analyzing common components of data pipelines and identifying limitations and bottlenecks that hinder scalability. It thoroughly examines existing data preparation methods, tools, technologies, and cutting-edge tools and methodologies such as Apache Nifi, Apache Atlas, and Apache Spark for addressing scalability challenges. The research draws insights from literature, industry practices, and state-of-the-art technologies to propose practical strategies and recommendations for designing a scalable data strategy in an enterprise setting. The study provides actionable insights and recommendations to enhance the performance of data pipelines in enterprise-grade data environments. The paper concludes with a summary of key findings, limitations, and future research directions, emphasizing the need for a well-designed data preparation pipeline that incorporates scalable data ingestion, efficient data transformation, and intelligent data storage strategies to ensure reliable and efficient data processing in enterprises dealing with large volumes of data.

CCS CONCEPTS • Data Pipeline • Data Governance • Data Lineage • Data Mesh • Data Profiling • Data Ingestion • Data Warehouse • Data Lake

Additional Keywords and Phrases: Data Pipeline, Data Governance, Data Lineage, Data Mesh, Data Profiling, ETL, ELT, Reverse ETL, Data Ingestion, Data Validation, Data Versioning

ACM Reference Format:
Sadig Akhund. 2023. Scalable Data Strategy for Enterprise-Grade Data Preparation Pipeline: A Scalability Optimization Study. 81 pages. https://doi.org/10.13140/RG.2.2.28382.72004

1 INTRODUCTION

In today's data-driven landscape, enterprises are grappling with formidable challenges in managing and processing massive amounts of data to extract meaningful insights and drive informed decision-making. Data preparation, a pivotal process that converts raw data into a usable format, plays a critical role in the data pipeline, as it significantly impacts the accuracy and reliability of downstream data analysis and modeling. However, traditional data preparation methods may struggle to keep pace with the ever-increasing volumes and complexity of data, resulting in scalability issues that lead to inefficiencies, delays, and suboptimal performance in the data pipeline. Hence, there is an urgent need to develop a scalable data strategy to bolster data preparation's efficiency and effectiveness in enterprise-grade data pipelines.

To address this need, this thesis presents a comprehensive scalability optimization study that seeks to analyze and optimize the data preparation process in enterprise-grade data pipelines. The study commences by meticulously analyzing the common components of data pipelines, and identifying the limitations and bottlenecks that impede scalability. This involves a comprehensive examination of existing data preparation methods, tools, and technologies that are currently prevalent in the industry. The study also delves into the challenges and complexities associated with large-scale data processing, including issues related to orchestration, data lineage, and data processing frameworks. Furthermore, the study explores cutting-edge tools and methodologies, such as Apache Nifi for orchestration, Apache Atlas for data lineage, and Apache Spark for large-scale data processing, to overcome the scalability challenges in data preparation. These tools and methodologies are thoroughly analyzed and evaluated to discern their strengths and weaknesses in the context of scalability. The research draws insights from a meticulous analysis of existing literature, industry practices, and state-of-the-art technologies to propose practical strategies and recommendations for designing a scalable data strategy in an enterprise setting. The ultimate goal is to provide actionable insights and recommendations that can significantly enhance the performance of data pipelines in enterprise-grade data environments. The organization of this thesis is as follows. To begin, an encompassing definition of common stages and subsections within a data pipeline will be outlined, providing the basis for analysis of the theories, industry methods, and contemporary technologies that tout the scalability of data preparation. Thereafter, intricate research from both literature and studies about data preparation pipelines and their scalability optimization will be reviewed and organized to assemble a theoretical framework upon which experimentation can take place. More precisely, the strategy and experimental setup employed in this study- including apparatuses and techniques needed for knowledge profiling, tracing, and performing evaluation- will be expounded upon. Clarification of rationale plus adjustments or modifications exercised due to desired goals set forth by the scaling innovating survey will also be specified. Founding itself in those facts, an assessment and examination of scaling development will be spoken of in-depth afterward; providing findings, understandings, and advisements derived from the study. The implications surrounding a scalable data approach in enterprise attractions will be extensively examined alongside any flaws or impediments picked up during the inquiry. Eventually, a wrapping-up of the main insights, limits, and prospective routes for subsequent examinations of data planning, data pipelines, and scaling progress will be the ending part. Summarization of results using the aforementioned constrictions shall be included as well as early ideas for future inquiries in the related field given.

Additionally, a thorough review of relevant literature and research on data pipelines, and scalability optimization will be presented to establish the theoretical framework for the study. Following the literature review, the methodology and experimental setup employed in this study will be detailed. This will include a description of the tools and techniques utilized for data profiling, data lineage, and performance evaluation. The rationale behind the selection of these methods will be provided, along with any modifications or adaptations made to suit the specific objectives of the scalability optimization study. Subsequently, the results and analysis of the scalability optimization study will be discussed in depth. This will involve presenting the findings, insights, and recommendations obtained from the analysis. The implications of these findings for designing a scalable data strategy in an enterprise setting

will be thoroughly discussed, along with any limitations or constraints encountered during the study. Finally, the paper will be concluded with a summary of the key findings, limitations, and future research directions. The conclusions drawn from the study will be summarized, and any limitations or constraints encountered during the research process will be acknowledged. Furthermore, suggestions for potential future research directions in the field of data preparation, data pipelines, and scalability optimization will be provided to guide further research in this area. The problem addressed in this research is the scalability of data preparation in enterprise-grade data pipelines. As organizations deal with increasing volumes of data, traditional data preparation methods may not be efficient or effective in handling the scale and complexity of the data involved. Data preparation, including acquisition, preprocessing, and validation is a critical step in the data pipeline that impacts the accuracy and reliability of downstream data analysis and modeling. Challenges and limitations associated with data preparation in enterprise-grade data pipelines may include processing delays, resource-intensive operations, data quality issues, and difficulties in handling large volumes of data, historical data, and real-time data. The need for scalability optimization arises from the demand for timely and accurate data-driven insights in enterprises, such as personalized recommendations and demand forecasting, to enhance customer experience and enable informed decision-making. Enterprises require a scalable data strategy that can efficiently handle large volumes of data, optimize data preparation processes, reduce processing times, and ensure data quality in an enterprise-grade data pipeline. Scalability optimization is crucial to overcome the challenges and limitations of data preparation in enterprise-grade data pipelines and enable organizations to unlock the full potential of their data assets.

A well-designed data preparation pipeline that incorporates scalable data ingestion, efficient data transformation, and intelligent data storage strategies is essential for enterprises dealing with large volumes of data, ensuring reliable and efficient data processing. To ensure that, data architects must follow the model presented below:

1. **Data Lineage System** - that is how data moves through their systems, identifies data sources and destinations, and tracks changes and transformations applied to data along the way.
    - Armed with clear data provenance.
2. **Data Mesh Architecture** - effective when there is more than one team who handles the data.
    - Where domains of each team are clearly defined, and data ownership is established
3. **Data Governance System** - that defines clear policies, establishes standardized processes, utilizes appropriate tools, fosters a data governance culture, and monitors performance.
    - Data Ownership - where clear accountability and responsibility for data assets are established
4. **Data Profiling Mechanism** - that analyzes, assesses, and documents the quality, accuracy, completeness, and consistency of data assets, to ensure they meet the defined data governance policies and standards, and helps organizations gain insights into their data, identify data quality issues, and make informed decisions about data management and data governance efforts.
5. **Managed Metadata Environment** - that provides a central repository for managing metadata, including data definitions, data lineage, data catalogs, and data dictionaries, to ensure consistency, accuracy, and traceability of metadata across the data pipeline.
6. **Data Processing Unit** - that is capable of handling both stream and batch processing of data, utilizing efficient data transformation techniques to process data in real-time or in batch mode, depending on the nature and requirements of the data being processed.
7. **Data Schema Blueprints** - these provide comprehensive and standardized documentation of the data schema or data model used in the pipeline, including tables, fields, relationships, constraints, and data types, to ensure consistency and accuracy in the structure and design of the data across different stages of the pipeline, and facilitate understanding and interpretation of the data by various stakeholders involved in the pipeline.
8. **Data Ingestion System** - that enables the ingestion of data from various sources into the pipeline, including ETL processes, data validation and enrichment, data integration, and data staging, ensuring that data is properly ingested and prepared for processing in the pipeline, and adhering to the defined data governance policies and standards. Data ingestion system plays a crucial role in ensuring the accuracy, integrity, and quality of data as

it enters the pipeline, and facilitates seamless data flow and processing throughout the pipeline, supporting its scalability and performance.

9. **Scalable Data Storage System** - that provides a scalable and efficient storage solution for the large volumes of data processed in the pipeline.
    - Deciding between a centralized data warehousing solution and independent data marts, or a combination of both, known as a dependent data mart system.
    - Selecting between a homogeneous or heterogeneous data warehouse architecture.
    - Establishing the topology or structure of a data lake
    - Determining the format in which data will be stored, which may involve specifying the data storage format such as Parquet, Avro, ORC, JSON, etc. Moreover, deciding on the appropriate compression and encoding techniques, however, most modern systems do this automatically.
10. **Efficient Data Versioning System** - that enables efficient and effective management of different versions of data assets within the pipeline, allowing for easy tracking and retrieval of historical data, comparison of changes over time, and proper handling of data updates and modifications.

It is believed that the implementation of these solutions ensures scalability across all steps of data preparation, including defining data requirements, selecting data sources, data extraction, data ingestion, data storage, and versioning. To support this statement, a case study based on a real-life scenario was conducted and described in the next section, and relevant open-source big data tools and solutions are introduced, along with the rationale for their selection.

The experiments in this research are based on certain presumptions and limitations. Their scope revolves around the scalability of data orchestration in enterprise-grade data pipelines and may not incorporate all viewpoints regarding data preparation or data pipeline optimization. The findings and recommendations of this research may be influenced by the specific context, tools, and technologies used in the study, and may not be directly applicable to all types of data pipelines or organizations. Furthermore, research can potentially be delimited by constraints such as time, resources, and availability of data. Consequently, it is noteworthy to recognize these assumptions and limitations while considering the outcomes of this study and determining the practical significance. Overall, this research contributes to the field of data preparation and data pipeline optimization by providing insights and recommendations for designing scalable systems. By addressing the challenges of data preparation scalability, this research aims to contribute to the advancement of data-driven decision-making and enable enterprises to harness the full potential of their data assets.

2 STEPS OF DATA PREPARATION

This chapter is organized as follows: The most common components of enterprise-grade pipelines, along with their respective substeps, are defined. A detailed description of each step is provided. Furthermore, technical details on how these steps can be further improved to enhance scalability are explored. Moreover, the reasons for the importance of these steps and how focusing on them can contribute to the overall goal are discussed. Additionally, the challenges that should not be neglected and key techniques and methods to address such issues are noted. Finally, the key aspects for each step that are included in the strategy for building a successful machine learning pipeline are defined. Understanding the significance of each step, the technical intricacies involved, and the challenges that may arise is essential for practitioners to build a robust and scalable pipeline. By incorporating appropriate techniques and methods, challenges can be overcome, and the pipeline can be optimized for improved performance and reliability.

A data pipeline is a series of interconnected stages or processes that are designed to extract, transform, and load (ETL) data from various sources, process it, and make it available for analysis, reporting, or other applications. Data pipelines are commonly used in machine learning and other data-intensive applications to manage the flow of data and ensure that it is properly prepared, validated, and transformed before being used in modeling or other downstream processes. The application areas of data

pipelines are vast and diverse, ranging from business intelligence, data analytics, and reporting to machine learning, predictive analytics, and artificial intelligence. In the context of machine learning, data pipelines are crucial for managing the end-to-end data flow, from data collection to model training and deployment. They play a significant role in ensuring that data used for machine learning is properly prepared, validated, and transformed to meet the requirements of machine learning algorithms and models. The difference that data pipelines make in machine learning and other data-intensive applications is immense. They enable organizations to efficiently manage the data flow, ensure data quality and consistency, and automate the data preparation and transformation processes. This helps in reducing manual errors, improving data accuracy, and ensuring that data used for machine learning is reliable and trustworthy. Data pipelines also enable organizations to scale their data processing capabilities and handle large volumes of data efficiently, which is essential for machine learning models that require substantial amounts of data for training and validation.

Based on extensive research and analysis of various pipelines utilized for diverse purposes and requirements, the findings of this study indicate that a data pipeline designed specifically for machine learning model training typically comprises three main stages: Data Preparation, Modelling, and Deployment. Each of these stages can be further subdivided into distinct steps, which collectively form a comprehensive framework for building an effective and efficient ML pipeline. The first stage, **Data Preparation**, involves the gathering, cleaning, and preprocessing of raw data to ensure its quality, integrity, and suitability for subsequent model training. This stage typically includes steps such as *data acquisition*, *data preprocessing*, and *data validation*. These steps are crucial in ensuring that the data used for model training is accurate, complete, and representative of the real-world scenarios that the model will encounter during inference. The second stage, **Modelling**, focuses on the development, training, and evaluation of machine learning models using the prepared data. This stage encompasses steps such as *model training*, *model evaluation*, and *model optimization*. Which involves selecting an appropriate algorithm or model architecture, partitioning data into training, validation, and test sets, training the model using various techniques such as supervised or unsupervised learning, and evaluating the model's performance using appropriate metrics. Additionally, this stage may involve hyperparameter tuning, model selection, and model interpretation to optimize the model's accuracy, generalization, and interpretability. The final stage, **Deployment**, is concerned with integrating the trained model into a production environment to make it available for real-world applications. This stage includes steps such as *model deployment*, *visualization*, *monitoring*, and *maintenance*. Which involve model deployment, model serving, and model monitoring. Model deployment involves deploying the trained model to a production environment, such as a cloud server or an edge device, and ensuring its operational readiness. Model serving involves exposing the model through APIs or other interfaces to accept and process incoming requests from applications or users. Model monitoring involves continuously monitoring the model's performance and detecting and addressing any anomalies or drifts in its behavior.

Furthermore, it is important to note that each stage of the pipeline entails its own specific set of tasks, requirements, or methods that must be meticulously implemented or executed to ensure the successful completion of substeps. As depicted in Figure 1, a comprehensive decomposition of each substep has been formulated, considering scalability. Notably, the primary focus of this paper lies in the Data Preparation stage, specifically in the domain of data acquisition. In the remainder of this paragraph, a concise overview of each task associated with the data acquisition stage will be provided, elucidating their significance in the context of building a robust and scalable data pipeline. The data acquisition step encompasses several critical tasks that are essential for the successful implementation of a data pipeline. These tasks include defining the data requirements, carefully selecting appropriate data sources, extracting relevant data from these sources, ingesting the extracted data into the pipeline, storing the acquired data in appropriate formats or databases, and maintaining proper versioning for data consistency and traceability. In the first task, defining data requirements, a thorough analysis is conducted to identify the specific data needs and characteristics that are relevant to the given use case or problem statement. This involves understanding the type of data required, its format, quality, and any

other specific attributes that are necessary for successful data processing. The next task is selecting data sources, where careful consideration is given to identifying the most suitable data sources that align with the defined data requirements. This involves evaluating various data sources such as databases, APIs, data warehouses, data lakes, or external data providers, and selecting those that are most relevant, reliable, and aligned with the purpose of the data pipeline. Once the data sources are identified, the data extraction process comes into play. This step involves extracting the relevant data from the selected data sources, which may include processes such as querying databases, scraping web data, or retrieving data from external APIs. Data extraction requires careful handling to ensure data accuracy, integrity, and security during the transfer process. After data extraction, the next task is a data ingestion, which involves loading the extracted data into the data pipeline for further processing. This step may involve converting data into the appropriate format or structure required for downstream data processing, such as transforming data into a standardized format or normalizing data values. Data storage is another crucial task in the data acquisition step, which involves storing the acquired data in appropriate storage systems or databases. This may include traditional relational databases, NoSQL databases, cloud-based storage solutions, or distributed file systems, depending on the nature of the data and the requirements of the data pipeline. Finally, versioning is an essential aspect of data acquisition, which involves maintaining proper version control for the acquired data. This ensures that changes to the data or data sources are appropriately tracked, and the data used in the pipeline is consistent and traceable over time. Versioning also allows for the reproducibility and repeatability of data pipeline processes, ensuring that the same results can be obtained with the same version of data. In conclusion, data acquisition is a critical stage in the data pipeline process that involves various tasks, including defining data requirements, selecting data sources, data extraction, data ingestion, data storage, and versioning. Proper execution of these tasks is essential to ensure the accuracy, reliability, and consistency of data used in the data pipeline and to facilitate downstream data processing for machine learning model training or other data-driven applications.

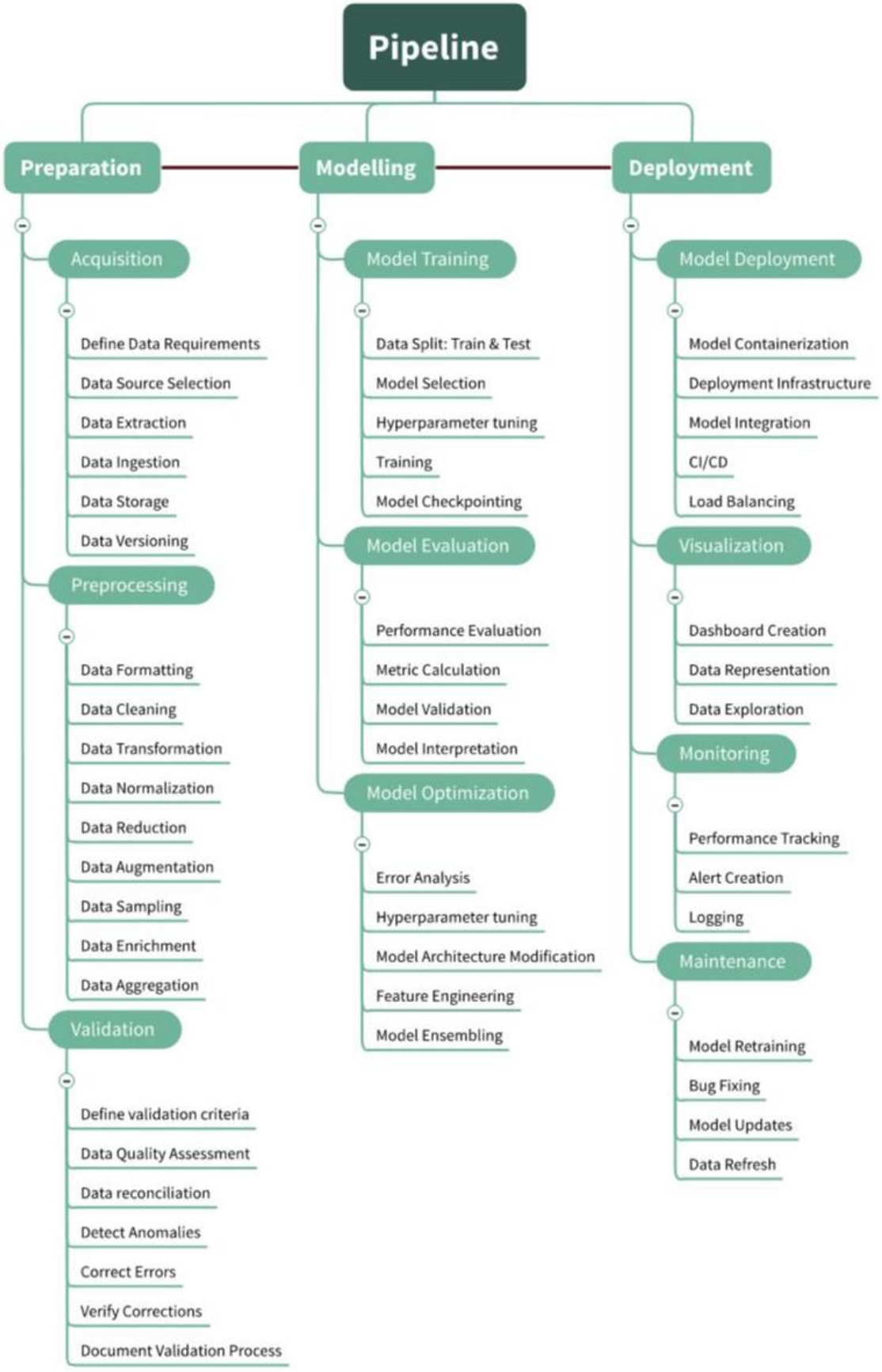

Figure 1: Flow Diagram Shows Three Stages of the Pipeline.

2.1 Defining Data Requirements

Dauntingly, big and varied data sets present a challenge to organizations so they can be handled with diligence. For the successful acquisition the information, maintaining quality, consistency, and allowance over an organizational range is significant due to its profound utilization in all aspects of the business. The procedure of determining data requirements is paramount in the foremost stage of acquisition since it sets the groundwork for the entire avalanche. It includes particularizing what data is needed, how exactly it should be collected, refined, then kept, and the necessary outcomes earned from it. To install the initial move in stipulating data prerequisites, recognizing the stakeholders who will put the data into use with their précised data needs is quintessential. This will aid in discovering any conflicting collisions or gaps in requirements and make sure that all stakeholders are on the same page concerning the demanded data and also why; thus, appeasing disagreements. Besides, it is very imperative to be cognizant of fluctuations in data requirements among different stakeholders which depend on their roles, responsibilities, and objectives set for each one. To tackle this issue, new approaches such as **data governance**, **data lineage**, and **data mesh** have been developed. These approaches have become popular because they help organizations define their data requirements, improve scalability, and guarantee the reliability and trustworthiness of the data. Therefore, in this discussion, we will delve into these concepts in detail and highlight their importance in enhancing the defining data requirements step of data acquisition. By the end of this discussion, you will have a clear understanding of how data governance, data lineage, and data mesh can help organizations acquire and manage their data more efficiently.

2.1.1Data Mesh Architecture

One relatively new concept called data mesh comes in handy when considering decentralization, autonomy, and domain-driven design. As defined by [1], The concept of Data Mesh, which was introduced by Zhamak Dehghani, proposes a distributed and decentralized approach to managing enterprise data. This approach considers each dataset as a distributed product, which is focused on specific domains. The aim is to establish a sense of data ownership and responsibility by assigning embedded engineers and product owners to manage the data and its accessibility to other teams. This differs from current data platforms, which are often centralized, monolithic, and rely on complex pipelines that lack ownership and responsibility at the data level. In essence, Data Mesh is a holistic approach that prioritizes domain-specific data management to enhance data ownership, responsibility, and accessibility across the organization. Figure 2 provides a visual representation of how data mesh architecture is different from traditional, centralized architecture.

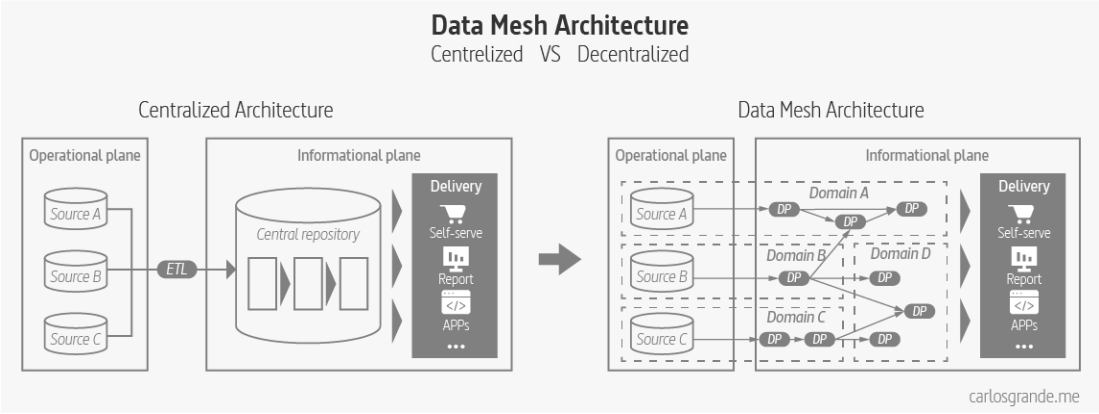

Figure 2: Decentralized vs Mech Architecture by C. Grande [9].

When building a sturdy pipeline, having pertinent data that satisfies the expectations of stakeholders is vital. By utilizing this method, organizations can authorize their data product teams to make informed decisions regarding how data is stored, collected, and utilized within their domain, bettering

its excellence and allowing more proficient exchange and collaboration between teams. At its core, data mesh involves creating small, self-organizing teams, known as data product teams, that are responsible for specific data domains. These teams are empowered to make decisions about how data is collected, stored, and used within their domain, based on the needs of their stakeholders. Data mesh also involves several technical practices, such as data ownership, data as a product, and federated data governance, that are designed to support the autonomy and flexibility of the data product teams. From a technical standpoint, data mesh is a technical approach that promotes autonomy and flexibility in data product teams. The approach involves practices like domain-driven design, APIs, and event-driven architectures to facilitate data sharing and collaboration among teams. Additionally, data observability plays an essential role in creating a shared understanding of data quality, lineage, and usage across the organization. Moreover, challenges in implementing a data mesh approach include ensuring team alignment with organizational goals and having technical infrastructure that supports autonomy and flexibility while also maintaining data security and compliance. To avoid scalability issues, organizations need to create well-defined and well-bounded data domains with clear ownership boundaries, data quality requirements, and effective communication channels. Furthermore, to optimize or automate this step, organizations can leverage automated data profiling and cataloging tools to streamline the process of data discovery and documentation. They can also use machine learning and other AI technologies to identify patterns and insights within the data, which can inform the development of more effective data requirements. Collaborative platforms and tools like data wikis and governance dashboards can facilitate communication and collaboration between data product teams.

According to the 'Data Mesh' book by O'Reilly [2], data mesh is based on *four* basic principles that form the foundation of its logical structure and operational model. These principles are intended to help organizations achieve the goals of data mesh, which include enhancing the value derived from data at scale, maintaining agility as the organization expands, and adapting to change in a constantly evolving and unpredictable business environment. Those principles are:

- **Principle of Domain Ownership –** Idea of distributing ownership of analytical data to the business areas that are closest to the data, whether they are the data source or the primary users [2].
- **Principle of Data as a Product –** By following this principle, domain-based data is shared directly with data users such as data analysts and scientists, as a product with a set of usability characteristics, including being discoverable, understandable, trustworthy, and natively accessible [2, 3]. This data product is defined by a set of data-sharing contracts, is autonomous with its lifecycle and model, and is encapsulated in a data quantum that includes all necessary structural components like metadata, code, policy, and infrastructure dependencies [2].
- **Principle of the Self-Serve Data Platform –** The Data Mesh addresses the limitation of the organization's ability to grow and expand its operations in utilizing data effectively, and as such, the platform must facilitate data sharing both within and outside the organization while also ensuring secure interoperability across multiple platforms. Without such interoperability, a single monolithic solution will limit scalability. Since data has limited value if it is not shared, the Data Mesh platform must be designed accordingly [3].
- **Principle of Federated Computational Governance –** As defined by Tartow, C., & Mott, A in their article at Starburst, data mesh emphasizes a move away from centralized data teams and architectures towards a decentralized model, which raises questions about governance functionality that remains under central IT organization and the responsibilities of domains. Although Data Mesh entails a shift from centralized to decentralized architecture, certain aspects of data governance need to remain under a central group to maintain balance and ensure comprehensive governance [1]. The principle involves establishing a data governance operating model that comprises a team consisting of domain representatives, data platform experts, and subject matter experts like legal, compliance, and security personnel. This model allows for

decentralized decision-making and accountability while promoting autonomy and agility within domains, as well as global interoperability of the Data Mesh. This incentivizes effective management of data governance, while balancing the autonomy of different domains with the need for effective collaboration across the organization [2].

In conclusion, data meshes have emerged as a powerful solution for managing data at scale. They provide a way to handle the complexity and heterogeneity of data in modern distributed systems by enabling a more decentralized approach to data management. Data meshes have a significant impact on scalability by enabling teams to manage data more efficiently and reducing the costs associated with centralization. Furthermore, data meshes indirectly improve machine learning performance by facilitating access to more diverse and high-quality data, which is essential for training more accurate and robust models. With the continued growth in data volume and complexity, data meshes are likely to play an increasingly important role in data management and machine learning in the future.

2.1.2 Data Governance

During the data acquisition phase, data governance is implemented through the establishment of policies, procedures, and standards to govern how data is obtained, organized, and used in an organization. These policies aim to ensure the accuracy, completeness, reliability, and compliance of data with relevant regulations and privacy laws. Organizations define data quality standards, data storage, and retrieval processes, and ensure ethical data collection with proper consent. Roles and responsibilities for data ownership, stewardship, and access are also defined. From a technical perspective, this involves implementing mechanisms such as data validation rules, automated data cleansing processes, and data profiling tools to ensure data quality. The accuracy and effectiveness of machine learning models depend greatly on the quality and integrity of the data used for training. Castra et al. provide a comprehensive analysis of the role of data governance in simplifying data management processes, as it is responsible for decision-making and accountability in data management [5]. However, this step presents challenges such as managing large and complex datasets, maintaining data accuracy and completeness, ensuring data privacy and security, and handling data from diverse sources and formats. Castro et al. propose ontology-based models that offer significant benefits for knowledge representation, particularly due to their formalization and expressive capabilities, in various fields, including data governance [5]. Ontology-based reasoning is an approach that uses ontologies to represent knowledge and perform automated reasoning on data [5,6,7]. This method is applied in data governance to simplify the management of large amounts of data by controlling decision-making and responsibilities related to data management [5,6]. The proposed data governance system in the paper is based on such reasoning, using semantic techniques and a shared knowledge plane to represent all data management processes. The system was tested in Telefonica's global video service, demonstrating its feasibility in reducing the complexity of managing big data environments [5]. In summary, ontologies enable the representation of behavior through rules, restrictions, and policies, providing a powerful means of structuring and organizing data, enhancing its usability, and facilitating decision-making processes. Furthermore, ontologies can be used to establish data governance policies that guide data management activities, ensuring consistent and structured data management. Overall, ontologies offer a robust and flexible framework for knowledge representation, which can be applied to various fields, including data governance, to improve data management practices and decision-making processes.

2.1.3 Data Lineage

Data lineage is a critical aspect of the data pipeline, as it tracks the origin, transformation, and movement of data throughout various stages of the process. Many people confuse data lineage with **data provenance**. For instance, Beneventano et al. define both terms, rightfully so, as "…where data came from, how it was derived, and how it was modified over time". Although similar, data lineage and provenance are related concepts but have different meanings. Data lineage refers to the process of

identifying the origins, transformations, and movements of data as it passes through various stages of a data pipeline. It involves understanding the source of the data, how it's collected, and how it's processed before being ingested into the pipeline. Data lineage is concerned with tracking the flow of data from its source to its destination and ensuring that the data is accurate, reliable, and aligned with the needs of stakeholders. Provenance, on the other hand, refers to the history of the data itself, including information about its origin, ownership, and any changes or modifications that have been made to it over time. Provenance is concerned with the metadata associated with the data and provides information about how the data was created, who created it, and how it has been used. Provenance is important for data governance, as it helps ensure that data is used appropriately and in compliance with organizational policies. In other words, data lineage is focused on the flow of data through a pipeline, while provenance is focused on the metadata associated with the data itself.

At the acquisition step, understanding the source of the data and how it is collected and processed before ingestion into the pipeline is vital. To realize this, it is imperative that metadata is captured, including *source*, *format*, and *transformation history*. Although crucial, data lineage presents a few challenges, including handling large volumes of data, data security, and privacy, and maintaining lineage as the pipeline evolves. However, well-defined data domains, clear ownership boundaries, data quality requirements, and standard definitions can mitigate these challenges. The graphical illustration in Figure 3 showcases the intricate data flow within a real-life data warehouse system, specifically highlighting the movement of information from tables and views toward the end goal of report generation, providing valuable insight into the underlying mechanics and processes of the system.

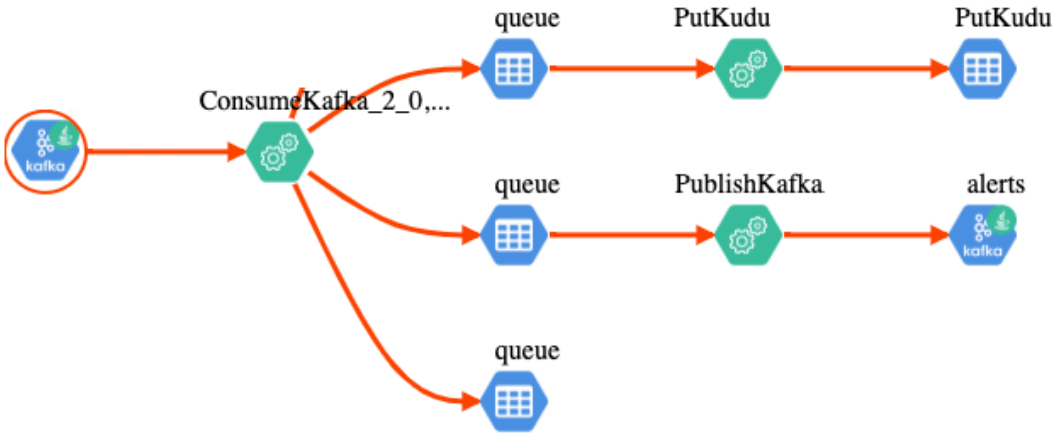

Figure 3: Apache Atlas. The lineage of Data Visualized. The movement of information from tables & views and the end goal of report generation are represented [10].

Devoting significant effort to this stage is of utmost importance when building a robust system as it ensures the *data's accuracy*, *reliability*, and *alignment with stakeholder needs*, leading to fewer errors, inconsistencies, and increased transparency and accountability. Organizations can automate the process by using data profiling and cataloging tools to streamline data discovery and documentation. Furthermore, machine learning and AI technologies can help identify patterns and insights to develop more effective data requirements. Finally, collaborative platforms and tools, such as data wikis and governance dashboards, can facilitate communication and collaboration between data product teams. The landscape of metadata management in ETL systems is constantly evolving, with a plethora of commercial tools available in the market. On one hand, ETL software products such as Informatica PowerCenter, IBM DataStage, and Microsoft SQL Server Integration Services include metadata repositories to store metadata generated and managed by the tool [10, 11]. Moreover, vendors now offer interfaces to import metadata from other tools, including databases and OLAP/reporting systems, into their repositories. Notably, the IBM Metadata Workbench, a component of the Information Server

product family, permits lineage and impact analyses of transformations implemented in the IBM DataStage tool [10]. However, lineage and impact analyses remain challenging to implement in Microsoft SQL Server Data Transformation Services [10, 12]. The metadata model that was once used to maintain transformations based on the Open Information Model (OIM) has been abandoned, and SQL Server now uses a similar approach to store transformation metadata, which can be accessed via an object-oriented API. Nonetheless, graph analysis methods are not provided out of the box, making lineage and impact analyses difficult to implement. On the other hand, specialized metadata management tools such as ASG Rochade and Adaptive Metadata Manager, are not tied to any specific ETL tool and can import metadata using standard CWM or specialized tool-specific interfaces [10]. This approach offers more flexibility in specifying a customized metadata model that caters to a company's specific requirements. However, these tools often use proprietary storage and querying approaches: one such example presented by Tomingas is that ASG Rochade uses a proprietary file-based storage queried internally using the Rochade [10] Procedural Language (RPL), while Adaptive Metadata Manager uses an Oracle database queried using SQL. Nevertheless, these commercial metadata management tools offer advanced mechanisms for data lineage and impact analyses, highlighting the potential for metadata management in the ETL field. Despite the limitations of commercial metadata management tools, they continue to advance with sophisticated mechanisms for data lineage and impact analyses. As mentioned in the introduction, the storage and querying approaches of these tools are often proprietary, making cross-repository queries from different vendors unfeasible. However, these tools offer promising opportunities for companies to efficiently manage their metadata and enhance their ETL systems.

From a technical perspective, data lineage involves capturing metadata about the data, including its source, format, and transformation history. This metadata can be stored in a metadata repository or data catalog, which can be queried to retrieve information about the data lineage. It may also involve the use of data profiling tools to identify any quality issues with the data and data governance tools to ensure compliance with organizational policies. One such ecosystem of tools can be identified as the **managed metadata environment** (MME). As depicted in Figure 4, the MME denotes the structural constituents and procedures that are necessary for the efficient and methodical accumulation, preservation, and distribution of metadata across the entire organization [13]. It encompasses the ideas of *metadata repositories*, *catalogs*, *data dictionaries*, and any other relevant term used for referring to the organized handling of metadata.

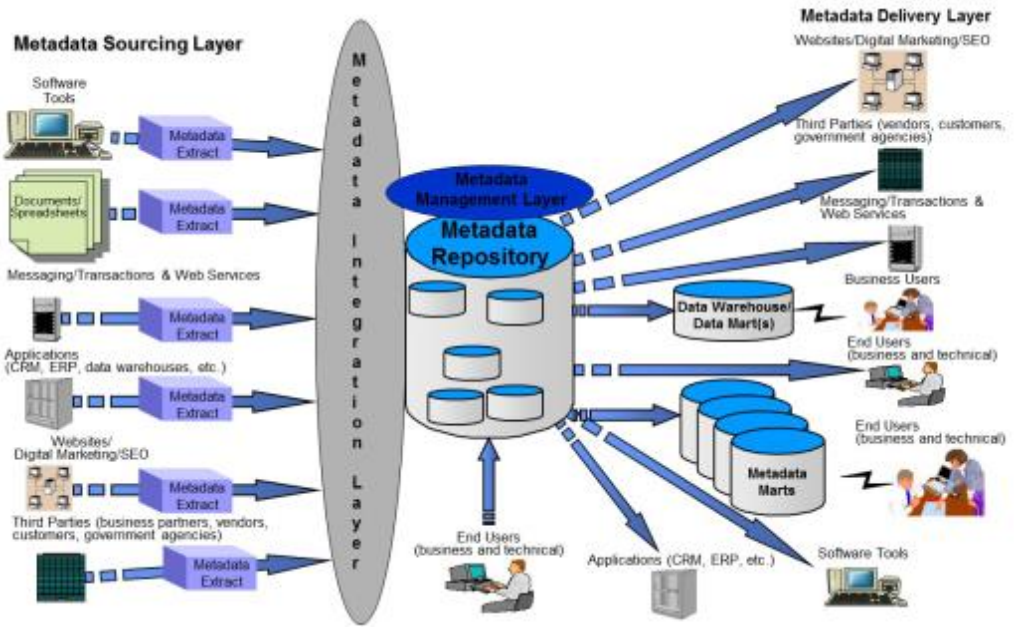

Figure 4: Managed Metadata Environment [13].

In conclusion, defining data requirements and understanding the data lineage is a critical step in building a robust system because it ensures that the data being collected and used is accurate, reliable, and aligned with the needs of the stakeholders. Therefore, focusing on this step helps to improve the quality of the data pipeline, reduces errors and inconsistencies, and increases the transparency and accountability of the data pipeline. Although some of the challenges of data lineage include dealing with complex data transformations, managing large volumes of data, and ensuring data security and privacy, it can also be difficult to maintain data lineage over time as the data pipeline evolves and new sources of data are added. Moreover, to avoid scalability problems, it's essential to create well-defined and well-bounded data domains that can be easily managed by the data product teams including creating clear ownership boundaries, defining data quality requirements, and establishing effective communication channels between the teams. It's also important to establish standard data definitions and ensure that the data is standardized and normalized across the organization. Essentially, if you were to optimize or automate this step several tools and technologies, such as data profiling and data cataloging tools, can help streamline the process of data discovery and documentation. Additionally, they can also use machine learning and other AI technologies to help identify patterns and insights within the data, which can inform the development of more effective data requirements. Finally, organizations can use collaborative platforms and tools, such as data wikis and data governance dashboards, to help facilitate communication and collaboration between the data product teams. In summary, the research presented here underscores the importance of addressing the obstacles that arise at this stage of the pipeline. To mitigate or deal with such challenges, organizations should consider implementing infrastructure such as cloud-based solutions or distributed databases, as well as utilizing data processing pipelines that can handle large volumes of data efficiently and leveraging automation tools to reduce the need for manual intervention, thereby increasing efficiency.

2.2 Selecting Data Sources

Picking data sources is a foundational step in creating a proficient, scalable data pipeline. It requires: recognizing which data to implement in the model and discovering those resources, either within an organization or outside it. Databases, warehouses, APIs, and third-party providers can all be sought after. The decision of which data sources to use should regard how useful, good, and available they are while also taking into consideration what the framework needs. Structuring and alteration may be needed before inclusion in the design. Technically, it begins by identifying the type of data being used –

structured or unstructured – and bringing together such data from databases, web services, or records. Afterward, that information must be adjusted into a format suitable for the scheme this process entails cleaning, filtering, and normalization. Once again, the validity of that data should always be checked; versioning can be valuable when managing advancements over time. Given the vast amounts of intricate details now accessible, considerable attention should be given to selecting data sources to avoid miscalculations and inappropriate judgment due to unjustified or inexact observations. Conscientiously choosing and picking available sources of data plays a critical role in the preparation period of the pipeline since it guarantees the reliability of the results set forth by the model. Nonetheless, overlooking the choice of resources at hand could bring about pricey blunders and defective preferences brought on by inadequate or incorrect findings. To avoid such situations, various techniques and methods can be employed, including **data exploration** and **profiling**, which involves performing exploratory data analysis to understand the data's characteristics, detect potential issues, and filter out irrelevant or low-quality data sources.

2.2.1 Data Profiling

Data profiling and data exploration are two important techniques used in the data preparation stage of the data pipeline. Both techniques involve analyzing and understanding the data before it is used in the model, but they differ in their focus and purpose. On one hand, data profiling is a process that involves analyzing the data to identify its *characteristics*, *structure*, and *quality*. It aims to create a summary of the data, such as its size, distribution, completeness, and consistency. Data profiling helps in identifying potential data quality issues and provides insight into the data's suitability for the model. On the other hand, data exploration involves digging deeper into the data to understand its *patterns*, *relationships*, and *correlations*. It aims to gain insights into the data and to identify any trends or patterns that can be used in the model. Data exploration may involve visualizing the data through charts, graphs, and histograms to identify patterns and relationships.

Data profiling is an intermediary stage in the data preparation phase of enterprise-grade pipelines, used to interpret the quality, structure, and content of data. A variety of procedural dimensions, such as statistical analysis, data pattern recognition, and data profiling rules, qualify researchers to acquire valuable knowledge of the specialties of their data set. In addition, a cornucopia of tools exists that streamlines the data profiling life cycle for increased efficiency and maximum performance. To achieve effective data profiling, various techniques can be employed, such as statistical analysis, data pattern recognition, and data profiling rules. These techniques provide insights into different aspects of the data, allowing organizations to assess and validate the quality of their data.

- **Statistical Analysis**: Statistical measurements, including mean, median, mode, standard deviation, and frequency distributions, proffer numerical understandings about the distribution, range, and versatility of the data values. These computations can guide researchers to identify outliers, discover tendencies, and assess the overall nature and durability of the information. For example, by weighing the standard deviation of the data values measures, analysts may detect prospective inconsistencies or variances, which would require further examination and rectifying inaccuracies.
- **Data Pattern Recognition**: Data pattern recognition follows identifying ceaseless patterns, embodiments, or structures in the collected bits of data emanating from phone numbers, email addresses, dates, or currency values. This modus operandi helps observers determine any probable error of facts quality by noting distinctions between observed forms versus predicted templates. To elaborate, evaluating the format of date values exposes any theoretical errors, like differing configurations (e.g., MM/DD/YYYY vs. DD/MM/YYYY), warranting additional clinical analysis and data reconciliation.
- **Data Profiling Rules**: Data profiling laws are preordained criteria or corporate directives established to judge the essence of data. Comprehended within such concepts include validating data kind, scope validation of data, range verification of figures, or other applicable aspects

contingent on the research query or area. Constructing and utilizing data profiling codes enables practitioners to systematically appraise their outlook against sanctioned standards for guaranteeing correctness and dependability. For instance, data format validation laws can check if data values are commensurate to the proposed data formation, for example ensuring phone numbers follow the habitual pattern (+994-XXX-XXX-XXXX) or that emails maintain the necessary framework (example@example.com).

Having a comprehensive understanding of data is crucial before using or processing it and data profiling is a technique that enables users to find metadata related to a dataset, providing basic information to aid in understanding it [14, 15]. It is a significant area of research for IT experts and scholars with many use cases, such as *data integration*, *data quality*, *data cleansing*, *big data analysis*, *database management*, and *query optimization* [14]. According to Liu et al. [14] some papers primarily focus on data profiling for relational data. Not only that, but also that other types of databases such as *time-series data*, *graph data*, and *heterogeneous data* also require data profiling. Moreover, as stated by Liu et al., data profiling tasks are classified into single-column data profiling, multiple columns data profiling, and dependencies. However, Naumann [16], on the other hand, classifies data profiling from single to multiple data sources. See Figure 5:

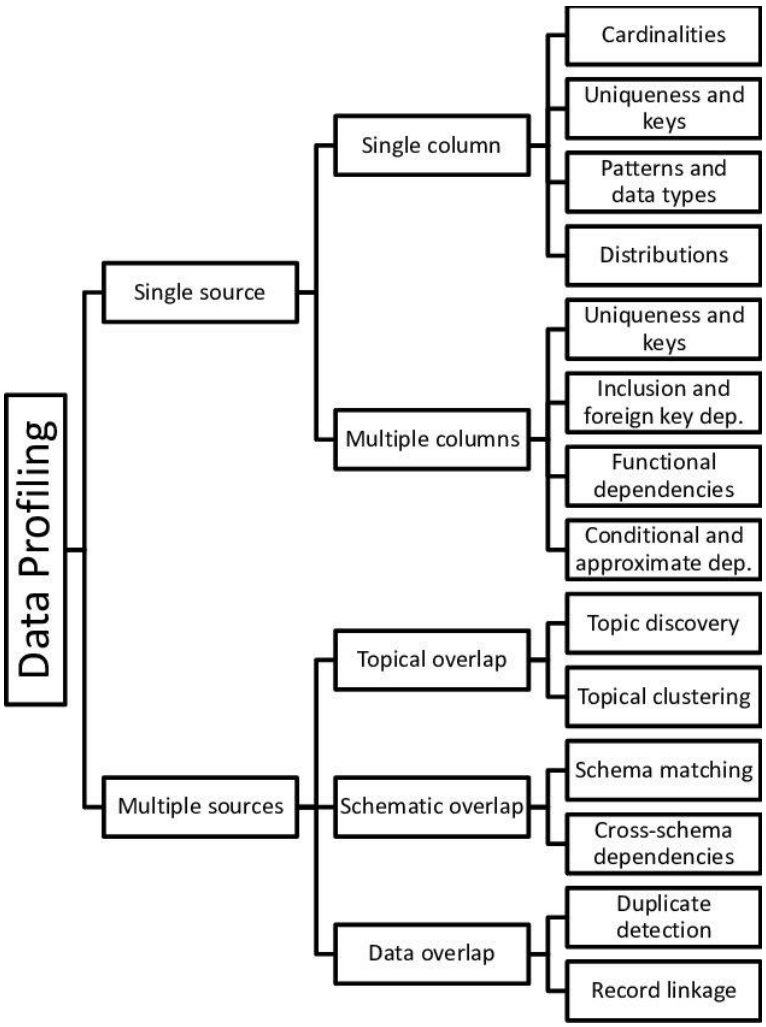

Figure 5: Data profiling tasks classified [16].

According to Figure 5, data profiling tasks are classified into two categories: "single sources" and "multiple sources" [16]. As indicated by Naumann, the tasks for "single sources" are well-established in

tooling and research, while the tasks for "multiple sources" represent new research directions for data profiling. To evaluate different methods and architectures for data profiling, a data profiling benchmark is necessary to define a set of tasks, data to be used, and efficiency measures. Creating realistic data is the most difficult part, and measures should go beyond traditional response time and cost efficiency evaluations to include approximations. The benchmark should evaluate both entire profiling systems and individual tasks. The computational complexity of data profiling presents three main challenges [14] – [16]:

- **Managing Input**: The results are computationally complex to discover and often require sorting steps.
- **Performing Computation**: The verification of complex constraints on all columns and combinations of columns in a database is necessary, which is often an exponential process.
- **Managing Output**: The datasets may be too large to fit into the main memory.

It is difficult to handle a large amount of data in traditional data mining algorithms, machine learning algorithms, and data profiling tasks, as it demands significant hardware resources and is time-consuming. To tackle these challenges, various tools and algorithms have been developed, such as relying on the capabilities of a DBMS, using innovative ways to handle the challenges, delivering only approximate results, and narrowing down the discovery process to certain columns or tables. The article by Liu [14] discusses two main strategies for data mining and analysis: **sampling** and **distributed systems**. In this paragraph, sampling will be the main focus. Sampling methods are an effective approach to reducing data volume and expediting data processing. In other words, it is a way to select a representative sample of data from a larger set to reduce the amount of data that needs to be processed. Sampling is particularly useful when computing power is limited, and approximate results are acceptable. Researchers must choose appropriate sampling methods and consider the biases that can result from sampling. Sampling involves probability and non-probability sampling, with probability sampling having every unit in a finite population having a certain probability of being selected. Moreover, it is used in various data applications, such as data profiling, data analysis, data mining, data visualization, and machine learning. There are different sampling techniques, including *simple random sampling*, *stratified sampling*, *systematic sampling*, *cluster sampling*, *oversampling* and *undersampling*, and *reservoir sampling*. In big data contexts, sampling can be performed with the help of big data computing frameworks such as MapReduce. Determining an effective sample size is crucial, since a sample size that is too small may result in incorrect conclusions, while a sample size that is too large may take too long to compute. Traditional methods for determining sample size have been summarized, and the most appropriate number of samples is when the accuracy rate reaches the maximum value. Additionally, **sampling error** occurs when a randomly chosen sample does not reflect the underlying population purely by chance while **sampling bias** occurs when the sample is not randomly chosen at all. Sampling bias is caused by the failure of the sampling design to truly extract the sample randomly from the population.

Single-Column Data Profiling

Single-column data profiling is the process of analyzing and understanding the characteristics and properties of individual columns or fields within a dataset. This type of data profiling focuses on a single attribute of the data and aims to identify patterns, anomalies, and statistics related to that attribute. Such profiling can involve various techniques, such as summary statistics, histograms, frequency distributions, and data quality checks. The goal is to gain insights into the data and identify any potential issues or discrepancies that may affect the quality or reliability of the data. For example, in a dataset containing customer information, single-column profiling may involve analyzing the "age" column to understand the age distribution of the customers, identify any outliers or missing values, and determine whether the data is consistent and accurate. This type of analysis can help data engineers and data scientists make informed decisions about the suitability of the data for their purposes and inform any

necessary data cleaning or preprocessing steps. According to Liu et al. [14], single-column data profiling tasks are classified into different categories such as *cardinalities*, *value distributions*, *patterns*, *data types*, and *domains*. Typical metadata resulting from these tasks are listed in Table 1. Moreover, for some tasks such as calculating the maximum number of decimals in numeric values, simple sampling methods may not be reliable. Additionally, identifying the domain of one column is often challenging and cannot be fully automated [14]. Among the different categories, cardinality, histograms, and quantiles are frequently used for query optimizers, and thus, sampling techniques are commonly used in these tasks.

Table 1: Single-Column Profiling Tasks [14].

Category	Task	Definition
Cardinalities	num-rows	Number of rows
	value length	Measurements of value lengths (minimum, maximum, median, and average)
	null values	Number or percentage of null values
	distinct	Number of distinct values; sometimes called "cardinality"
	uniqueness	Number of distinct values divided by the number of rows
Value Distributions	histogram	Frequency histograms (equi-width, equi-depth, etc.)
	constancy	Frequency of the most frequent value divided by the number of rows
	first digit	Distribution of the first digit in numeric values
	quartiles	Three points that divide the (numeric) values into four equal groups
Patterns, Data Types and Domains	basic type	Generic data types, such as numeric, alphabetic, alphanumeric, date, time
	data type	Concrete DBMS-specific data types, such as varchar, and timestamp.
	size	Maximum number of digits in numeric values
	patterns	Histogram of value patterns
	domain	Classification of the semantic domain, such as credit card, first name, city, phenotype
	decimals	Maximum number of decimals in numeric values
	data class	Semantic, generic data types, such as code, indicator, text, date/time, quantity, identifier

[a] Typical metadata resulting from tasks.

In summary, selecting the right data sources can minimize the risk of overfitting or underfitting the model, increasing its ability to generalize to new data. Prioritizing the selection of data sources can also help build a scalable and flexible model that can handle changing data sources or requirements. This can future-proof the system, making it easier to maintain and update. Furthermore, data profiling focuses on creating a summary of the data, while data exploration focuses on discovering insights and patterns in the data. Both techniques are important in the data preparation stage and can help in selecting appropriate data sources, identifying potential data quality issues, and gaining insights into the data that can be used in the model. Moreover, big data is susceptible to selection bias, and many scholars study how to solve selection bias in the sampling process. Additionally, there are several challenges associated with selecting data sources for machine learning models. Some of these challenges include identifying the most relevant and useful data sources, ensuring the quality and accuracy of the data, managing data from multiple sources, and dealing with data that may be incomplete or inconsistent. Also, selecting data sources can be time-consuming and may require specialized knowledge or expertise to identify and extract the necessary data. The challenges of selecting data sources can vary depending on the specific project or use case, but some common challenges include:

- **Finding relevant data sources**: It can be difficult to locate and identify data sources that are relevant to the problem or use case at hand. This may involve searching for data across multiple systems or platforms or working with third-party providers to obtain the necessary data.
- **Ensuring data quality**: The quality of data can have a significant impact on the performance of the machine learning model. It is important to ensure that the data is accurate, complete, and consistent and that any missing or erroneous data is appropriately handled.

Managing data compatibility: Different data sources may have varying formats or structures, which can make it difficult to integrate them into a single model. It is important to consider how the data will be transformed, cleaned, and processed to ensure compatibility with the model.

- **Addressing data privacy concerns**: Some data sources may contain sensitive or personally identifiable information, which can raise privacy concerns. It is important to ensure that the data is collected, stored, and processed in compliance with applicable privacy regulations.

To avoid problems with scalability when selecting data sources, several key points should be considered. First, it is important to identify data sources that are easily scalable and can handle large volumes of data. This may involve selecting cloud-based data sources or using distributed storage systems like Hadoop or Spark. Second, data should be organized and stored in a way that allows for efficient processing and retrieval. This may involve using data partitioning or indexing techniques to optimize data access. Finally, it is important to consider the computational resources required to process the data and ensure that there is enough capacity to handle growing data volumes over time. This may involve using parallel processing techniques or scaling up the computational resources as needed. In conclusion, selecting data sources is crucial to building a robust machine learning system that produces accurate and reliable results, minimizing errors or biases, and future-proofing the system for scalability and flexibility. By carefully considering and selecting data sources, we can improve data quality and relevance, reduce risk, and build a system that can adapt to changing data sources and requirements.

2.3 Data Extraction

Data extraction is an indispensable part of data preparation, necessitating the collection and retrieval of raw numbers from multiple sources. To extract data, relevant sources are identified, and a connection to the source is established. To build a connection to the source of data, suitable methods such as ODBC, JDBC, or even REST API are utilized. Depending on the source, the retrieval procedure may involve querying a database, parsing a file, or sending a request to an API endpoint. For file-based data sources, extraction may involve parsing text files or using specialized libraries for specific file formats. Once the data is extracted, it can be transformed and pre-processed to meet the data requirements of the model. The compilation process of data extraction involves some discrete steps, exclusive to this stage. Primarily, there is the recognition of sources and the specific data that needs to be extracted. Afterward, the fit system for gathering the data shall be decided, and the pattern/structure of the extracted data established. At long last, the desired data will be extricated using the opted approach. The next segments will further explore the previously mentioned phases in greater depth.

2.3.1 Determining the Data Sources

According to Talend [24], the term "data source" can refer to the original location where data is created or where physical information is first converted into a digital format. However, even highly refined data can serve as a source as long as it is accessed and used by another process. Examples of data sources include *databases*, *flat files*, *live measurements* from physical devices (IOT devices), *scraped web data* (web crawlers), and *various static and streaming data services* (APIs) available on the internet. That said, databases are the most commonly used data sources and are found in relational database management systems (RDBMS). In this context, the Data Source Name (DSN) is an essential concept [24]. A DSN is a pointer within a destination database or application that indicates the actual data location, whether it is local or remote, and whether it is in a physical or virtual location. The DSN is not necessarily the same as the relevant database or file name, but it is a label or address used to access the data. Furthermore, the interpretation of the term "data source" is differentiated by Talend depending on the systems used to ingest data as following examples [24]. For instance, in the Java software platform, a "Datasource" is a specific object that represents a connection to a database, while in newer platforms, "Datasource" refers to any collection of data that provides a standardized way of access. It is essential to understand the context of the systems used to interpret the meaning of data source terminology. Furthermore, data sources play a significant role in the transportation and management of

data. Different network protocols, such as FTP, HTTP, and APIs, are used to move data from various services and applications [25]. For instance, Adobe Analytics uses a file data source uploaded through FTP to process data automatically. When it comes to security, SFTP and FTPS with added TLS encryption are employed to protect sensitive information [24]. In addition to network protocols, APIs also provide more customized access methods for managing data sources. Spark, for example, offers an API with different implementations for generic relational sources and detailed JDBC connections. Other protocols, such as NFS, SMB, SOAP, REST, and WebDAV, are also used to transfer data within APIs or as standalone processes [24]. The primary objective of data sources is to make it easier for users and applications to connect and transport data to its intended destination. They collate essential technical information and present it in a user-friendly manner, saving stakeholders from dealing with low-level connection details. Moreover, data sources provide consistent storage of connection information, which makes processes such as system migrations or structural changes much smoother. Overall, data sources are critical for seamlessly integrating disparate systems and ensuring that data consumers can focus on processing and utilizing data to its fullest potential.

2.3.2 Data Ownership

When determining data sources, it is also important to consider **data ownership** because it affects the accessibility and availability of the data. The owner of the data may have specific requirements for how the data should be handled, including how it should be extracted, where it can be stored, and who can access it. Therefore, understanding data ownership is an important aspect of data extraction, as it helps ensure that data is being extracted in a way that aligns with the owner's policies and requirements. In the world of data management, it is important to determine who is responsible for keeping data accurate and up-to-date [22]. This is what is meant by data ownership. During the data design phase, it is crucial to identify who can write data to the directory. There are various approaches to determining data ownership, such as allowing only a small group of directory content managers to have read-write access to the directory while the rest of the users have read-only access. Alternatively, individual users can be given the authority to manage certain subsets of information, such as their passwords or personal information. A person's manager can also be given the authority to write to specific subsets of their information, such as contact information or job title. Additionally, organization administrators can create and manage entries for their organization, becoming directory content managers. To provide access privileges to specific groups of people, roles can be created for each group, such as human resources, finance, or accounting. These roles can be given read or write access to the data their respective groups need, such as salary information, government identification number, and home phone numbers and address. Overall, determining data ownership is essential in ensuring the accuracy and reliability of data in the directory.

Data ownership is a critical aspect of managing data in any organization. It ensures that data is well-governed, protected, and effectively utilized [23]. Assigning ownership to data assets is important for a variety of reasons. Firstly, it provides a clear line of responsibility for protecting the data and ensuring its quality [23]. By designating an owner for a particular data asset, it becomes the responsibility of that person to ensure that the asset is secure, reliable, and accessible to the appropriate stakeholders. This includes ensuring that the data is properly structured, stored, and managed according to industry best practices and regulatory requirements. Another key benefit of data ownership is that it promotes easier collaboration across teams [23]. Having a designated owner for a data asset eliminates the need for time-consuming and often confusing communication channels to track down the person with the most domain knowledge of an asset. The asset owner is identified explicitly, making it easy for other team members to know whom to reach out to for questions, updates, or any issues related to the data asset. This creates an instant path of communication between team members and the asset owner, improving data quality and ensuring data is being used effectively across the organization. Finally, assigning data ownership promotes accountability within the organization [23]. When assets don't have owners, data quality issues such as outdated data pipelines, irrelevant logic, and duplicated data can pile up, and no one is held

responsible for them. By assigning an owner, the organization ensures that one person is held accountable for maintaining the quality of the asset. This person will be responsible for ensuring that the data is refreshed regularly, cleaning up any duplicates, and ensuring that the data asset remains relevant and accurate. This provides a clear chain of responsibility and eliminates the guesswork involved in determining who should be maintaining what. In conclusion, data ownership is crucial in promoting effective data governance, collaboration, and accountability in any organization. By assigning ownership to data assets, organizations can ensure that their data is secure, reliable, and accessible to the appropriate stakeholders. Data ownership also promotes easier collaboration among team members and ensures that data quality issues are promptly addressed by assigning clear responsibility. As data becomes an increasingly valuable resource for organizations, assigning data ownership will become more important in ensuring that organizations effectively manage their data assets.

2.3.3 Determining Extraction Method

Determining the data extraction method is a crucial step in the data extraction process. It involves selecting the most appropriate method to extract data from the source systems while considering the data sources and the data extraction requirements. The data extraction method can vary depending on the type of source system and the type of data being extracted. For example, if the source system is a relational database, the extraction method may involve running SQL queries to retrieve data from tables. On the other hand, if the source system is a web-based application, the extraction method may involve using web scraping tools to extract data from HTML pages. One way for organizations to collect and store information from different sources is by using a cloud warehouse as a central data store. To achieve this, they typically use the Extract, Transform, and Load (ETL) process, which involves several steps. The first step is data extraction, where data is obtained from various sources through APIs or webhooks and then staged into files or relational databases. The second step is data transformation, where the data is converted into a format that's appropriate for reporting and analytics. This step involves enriching and validating the data, applying business rules, and ensuring consistency across all data fields. Finally, the high-quality, transformed data is loaded into a target data store such as a data lake or a data warehouse, where it can be accessed by other stakeholders for analysis and reporting purposes. ETL is a widely used approach across different industries and organizational sizes to integrate various applications, data services, and unstructured files. A well-engineered ETL pipeline with an appropriate data extraction process can provide valuable insights and ensure that stakeholders have access to complete information, enabling them to make informed decisions and eliminating ambiguity resulting from incomplete or uncertain data.

While data extraction is a crucial component of data analysis, it comes with its own set of challenges. These challenges include *managing data volume*, *considering data source* and *API limitations*, *data complexity*, *security and privacy*, and *monitoring data extraction processes* [25]. Firstly, data volume management can be challenging if data extraction tools are designed for smaller amounts of data, and parallel extraction solutions may be necessary when it comes to handling larger quantities. However, designing and maintaining these solutions can be complex. Consequently, for large volumes of data, it may be necessary to use a distributed data processing system such as Apache Spark or Apache Hadoop to extract data in parallel. Next up on the list are data source or API constraints must also be considered, as data sources vary and have different extractable fields [25]. APIs and webhooks may have restrictions on the amount of data that can be extracted at once. Another factor to consider is the data complexity and structure. If the data is structured and well-defined, then a straightforward extraction method such as SQL queries may suffice. However, if the data is unstructured or semi-structured, then more advanced extraction methods such as web scraping or natural language processing may be required. Moreover, data security and compliance requirements are also essential factors to consider when selecting the data extraction method. It is important to ensure that the method does not violate any data privacy regulations or compromise the security of the source systems. To provide a concise summary of the aforementioned points, it can be concluded that, determining the data extraction method involves selecting the most

appropriate method to extract data from the source systems while considering the data sources and the data extraction requirements and selecting the right data extraction method, organizations can ensure that they can extract the necessary data accurately and efficiently while adhering to regulatory and security standards.

2.3.4 Defining Schema

Defining the data schema is an important step in the data extraction process, as it establishes the structure and format of the data that will be extracted. A data schema is a blueprint or plan that defines the data types, attributes, relationships, and constraints that govern the data. To define the data schema, it is necessary to have a clear understanding of the data sources and the requirements of the target system or application. This involves identifying the data entities and their attributes, as well as any relationships or constraints between them. The schema can be defined using a variety of tools and techniques, including data modeling languages such as Entity-Relationship (ER) diagrams or Unified Modeling Language (UML) diagrams. These tools can help visualize the structure of the data and the relationships between different data entities. The data schema should be designed to meet the specific needs of the target system or application, while also ensuring data integrity and consistency. This involves defining data types and constraints that reflect the semantics of the data and ensuring that the data is valid and consistent. Once the data schema has been defined, it can be used to guide the data extraction process, ensuring that the extracted data is in the expected format and structure. It can also be used to validate the extracted data and ensure that it meets the requirements of the target system or application. In summary, defining the data schema is an important step in the data extraction process, as it establishes the structure and format of the data that will be extracted. This involves identifying the data entities and their attributes, as well as any relationships or constraints between them, and designing the schema to meet the specific needs of the target system or application. The data schema can be defined using a variety of tools and techniques and can be used to guide the data extraction process and validate the extracted data.

2.4 Data Ingestion

Data Ingestion is a critical step in the process of acquiring data from various sources and loading it into a data storage system for further processing. This step involves identifying the data sources and their formats, connecting to them, extracting the data, and transforming it into a format suitable for storage and processing. Downstream applications can then access the extracted data that has been loaded into a data storage system such as a database, data warehouse, or data lake. To achieve this, Data Ingestion requires a range of tasks such as defining data sources and formats, establishing connectivity to the sources, implementing data extraction logic, and transforming data into a standardized format. The use of tools and technologies such as ETL (Extract-Transform-Load) frameworks can automate these tasks. The data can be ingested in batches or streams, depending on the data source and processing requirements. Data Ingestion plays a crucial role in building a scalable and robust machine learning system by ensuring reliable data collection, transformation into a consistent format, and availability for processing. By focusing on this step, data quality issues such as missing or inconsistent data can be identified and addressed. Moreover, proper labeling and tagging of data for downstream processing can be ensured. A well-designed data ingestion process can minimize data loss and enable timely and efficient data processing. Moreover, this step can be a complex and challenging process, particularly when dealing with large and diverse data sources; data sources may have different formats, structures, and quality levels, leading to inconsistencies and errors in the ingestion process. Additionally, connectivity and access issues can also arise, especially when dealing with secure or remote data sources. Furthermore, the high volume of data involved can lead to scalability issues if not handled properly. There are two common methods of data ingestion: real-time and batch-based. The following subchapters delve into more details about these methods, as well as the most common ingestion processes like extract-transform-load (ETL), extract-load-transform (ELT), and Reverse ETL.

2.4.1 Real-Time or Stream Ingestion

Streaming data ingestion refers to the process of continuously collecting and analyzing data in real time from various sources, such as IoT sensors, log files, clickstreams, and message-based transactions. This approach allows for immediate processing and analysis of events, enabling businesses to quickly respond to time-sensitive opportunities. Similarly defined by the Airops team [27], real-time data ingestion (a.k.a. streaming or stream processing) involves collecting, manipulating, and loading data as soon as it is generated to create a continuous output. This approach is ideal for organizations that analyze data from web, mobile, and server events, and those that require immediate, reactive actions. Real-time data ingestion is also useful for monitoring IT systems, manufacturing equipment, and Internet of Things (IoT) devices. For instance, a SaaS business that provides a tracking app for on-demand delivery would require real-time data ingestion. Top streaming analytics technologies include Apache Kafka, Amazon Kinesis, and Confluent. To prevent overwhelming applications with high message volumes, micro-batching can be used to collect messages and provide them to consuming applications at regular intervals. Alternatively, if the data source is a traditional data file, traditional batch ingestion can be employed. Real-time data stream processing has numerous applications, such as sentiment analysis of social media streams to quickly react to changes in customer perceptions. For instance, retailers can collect and process real-time feeds from in-store beacon systems to identify potential customers who have shown interest in a particular product and are near a physical store. In response, personalized offers can be sent via SMS or email within seconds to convert them into customers. Sales and marketing systems can also utilize clickstream data to trigger interactions with chatbots or agents, enhancing customer engagement and driving sales.

2.4.2 Batch-Based Ingestion

On the other hand, batch processing is the most widely used form of data ingestion and is preferred when real-time data is not required [27]. This approach enables organizations to collect data in large quantities at specific intervals or during a scheduled event. Batch processing is more affordable and is generally the preferred option for most businesses. Batch-based data ingestion is commonly used in data analytics work such as business intelligence, data science, and machine learning. However, it's not uncommon for organizations to use both real-time and batch-based data ingestion. When deciding which method to use, it's crucial to establish your organization's standard of what "real-time" means, as most companies that require real-time ingestion want "near real-time" ingestion, which is a batch-based process. Using real-time data ingestion sparingly is recommended as it is costlier and more complex. Remember that infrastructure costs can vary greatly depending on the difference between processing times, and batch processing is generally recommended for anything over five minutes.

2.4.3 Preprocessing with ETL

The evolution of data management techniques has been shaped by the existence of databases past couple of decades. Originally developed during the era of on-premises servers and hardware-based solutions, ETL (Extract, Transform, Load) emerged as a process to manage data from multiple sources, reduce storage costs, and enable meaningful insights for business analysts writing SQL. While the core principles of ETL remain unchanged, significant advancements have been made in storage and processing technologies, notably with the rise of Cloud Data Warehouses. Today, companies increasingly leverage cloud-hosted data warehouses provided by Amazon, Google, Microsoft, and others, eliminating the need for on-site data storage, hardware management, and scaling concerns. Cloud-based solutions offer the flexibility to scale storage and computation resources to virtually infinite levels, alleviating physical constraints and simplifying data stack setup. As a result, the traditional ETL approach has evolved, consequently giving rise to ELT (Extract, Load, Transform) and Reverse ETL.

In academic language, an **ETL** pipeline is a series of interconnected processes used to transfer data from one or multiple sources to a database, typically a data warehouse. The acronym ETL stands for "extract, transform, load," which are the three essential steps in data integration to move data from one

database to another. In ETL, data is first extracted from various sources, such as databases, APIs, or external systems. Then, the data is transformed or processed to clean, enrich, aggregate, or format it according to the requirements of the target system or application. Finally, the transformed data is loaded into a data warehouse or data mart for storage and analysis. The loaded data can then be utilized for reporting, analysis, and generating actionable business insights. The benefits of implementing an ETL pipeline are significant in the context of preparing data for analytics and business intelligence. Source data from various systems, such as CRMs, social media platforms, and web reporting, needs to be consolidated and transformed to fit the parameters and functions of the destination database to provide valuable insights. An ETL pipeline facilitates this process by centralizing and standardizing data, making it easily accessible to analysts and decision-makers, while also freeing up developers from technical implementation tasks, allowing them to focus on more strategic work. Additionally, ETL pipelines are commonly used for data migration from legacy systems to a data warehouse and for enabling deeper analytics beyond basic transformations. Characteristics of an effective ETL pipeline include providing *continuous data processing*, being *elastic* and *agile* to adapt to changing data requirements, utilizing isolated and independent processing resources, increasing data access for decision-makers, and being easy to set up and maintain. With the shift towards cloud-based software services and advancements in ETL pipelines, organizations now have the opportunity to simplify their data processing and implement continuous processing methodologies without disrupting existing processes. This can be done incrementally and strategically, starting with specific types of data or areas of the business, rather than undergoing costly rip-and-replace approaches.

ELT, which stands for "extract, load, and transform," refers to the processes involved in replicating data from a source system to a target system, such as a cloud data warehouse. In the extraction step, data is copied from the source system, followed by the loading step where data is replicated into the target system. Once the data is in the target system, organizations can apply necessary transformations to suit their specific needs or requirements, which is the transformation step. ELT is considered a modern variation of the traditional ETL process, where transformations occur before the data is loaded, resulting in a more complex data replication process. One significant difference between ETL and ELT is the infrastructure used for running transformations. ETL tools typically require processing engines to perform transformations before loading data into the destination system. However, with ELT, organizations can utilize the processing engines within the destination system itself, streamlining the data-loading process by eliminating an intermediate step. ELT is especially suitable when the destination system is a cloud-native data warehouse, such as Amazon Redshift, Google BigQuery, Snowflake, or Microsoft Azure SQL Data Warehouse. In such cases, organizations can transform their raw data as needed, when needed, and as per their specific use case, instead of performing transformations as a step in the data pipeline. There are several business benefits of adopting ELT and utilizing cloud-based data warehouses and data lakes. Firstly, ELT generally provides a faster time to value, as it eliminates the time-intensive and resource-heavy transformation step before data loading or integration, making business intelligence available more quickly. Secondly, ELT enables scalability, as cloud platforms allow for almost unlimited scaling within seconds or minutes, unlike on-premises data warehouses that require ordering, installation, and configuration of new hardware. Lastly, ELT offers flexibility, as many BI tools allow schema determination on read and on-demand transformations, enabling organizations to replicate raw data into their data warehouse or data lake and apply transformations as and when needed. This flexibility allows organizations to adapt their data transformations to their changing business needs, making ELT a versatile approach for data replication and processing in modern data architectures. In conclusion, ELT, with its focus on extracting, loading, and transforming data in the target system, offers several advantages over traditional ETL processes, particularly in conjunction with cloud-based data warehouses and data lakes, including faster time to value, scalability, and flexibility. By leveraging the processing capabilities of destination systems, organizations can efficiently replicate and transform their data to meet their specific business requirements.

Reverse ETL, which involves copying data from a central data warehouse to operational systems and SaaS tools, enables business teams to leverage data for driving actions and personalizing customer experiences. Unlike traditional ETL, which extracts, transforms, and loads data into a data warehouse, Reverse ETL syncs data from the warehouse to operational systems such as marketing automation tools or ad platforms. While both methods use batch processing, there are key technical differences between ETL and Reverse ETL. ETL is primarily used for analytics use cases, consolidating data from various sources into a unified view, and allows for easy deletion and re-ingestion of data if mistakes occur. On the other hand, Reverse ETL syncs rows of data from the warehouse to internal systems, and requires careful handling of data structure restrictions and avoiding accidental data overwriting, as operational tools usually lack undo or rollback functionalities. Furthermore, Reverse ETL requires deduplication and comparison of current data with previously synced data to avoid rate limits and sync failures. Data warehouses offer more flexibility in this regard compared to operational systems. In summary, ETL focuses on data integration and getting data into the warehouse, while Reverse ETL focuses on Data Activation and getting data out of the warehouse to operational systems. In summary, Reverse ETL is a valuable process that unlocks the potential of data stored in data warehouses by syncing it to operational systems and SaaS tools for proactive use by business teams. It differs from traditional ETL in terms of data flow direction, data structure restrictions, and handling of data updates. By understanding the differences between ETL and Reverse ETL, businesses can effectively utilize both methods to extract, transform, and load data for various use cases, enabling data-driven decision-making and personalized customer experiences.

In conclusion, the evolution of data management techniques, from traditional ETL to ELT and Reverse ETL, has been driven by advancements in storage and processing technologies, particularly with the rise of cloud-based data warehouses. Cloud-hosted solutions offered by Amazon, Google, Microsoft, and others have eliminated the need for on-premises data storage, hardware management, and scaling concerns, providing organizations with scalability, flexibility, and faster time to value. ELT, with its focus on extracting, loading, and transforming data in the target system, allows for more efficient data replication and processing, leveraging the processing capabilities of destination systems. This results in faster time to value, scalability, and flexibility, making it an attractive approach for data integration in modern data architectures. Additionally, Reverse ETL, which syncs data from a central data warehouse to operational systems and SaaS tools, enables organizations to leverage data for driving actions and personalizing customer experiences. With the ability to replicate and transform data as needed, Reverse ETL empowers business teams to make data-driven decisions and deliver better customer experiences. Overall, these advancements in data management techniques, along with cloud-based solutions, have transformed the way organizations handle data, providing them with more agility, scalability, and flexibility to meet their evolving business needs.

2.4.4 Scalability Challenge

To ensure the scalability of a data ingestion process, it is crucial to consider various factors such as data volume, data velocity, and data variety. These factors play a crucial role in determining the most suitable data ingestion approach and the ideal data storage system, which could be either a data warehouse or a data lake. Additionally, factors such as data quality, consistency, access, and security must also be taken into account. When it comes to determining the right data ingestion approach, several factors must be carefully considered, such as the data volume, latency requirements, processing complexity, and the need for real-time insights. The next steps may assist in avoiding discrepancies while designing. First, when identifying the sources of data, it is important to consider their reliability and accuracy. Additionally, examining the source systems and applications that produce the data can provide insight into the data's context and meaning. Then, analyzing the processing requirements should also include an assessment of the tools and technologies required for data processing and analysis, as well as the level of expertise and resources available for implementation [25, 27]. This can help identify potential constraints and limitations in the data processing approach. Third, in determining the latency

requirements, it is important to consider the timeliness of insights and whether the real-time analysis is necessary [26, 27]. Depending on the application, the real-time analysis may be critical for identifying trends and patterns that could otherwise be missed. Subsequently, evaluating the trade-offs between batch processing and streaming should also take into account factors such as *data complexity*, *processing needs*, and *infrastructure constraints*. For example, batch processing may be more appropriate for large, complex data sets that require significant processing power, while streaming may be better suited for real-time, event-driven data processing. Finally, choosing the appropriate data processing approach is critical for ensuring the *accuracy*, *reliability*, and *timeliness of insights* [27]. It is important to regularly monitor and evaluate the data processing approach to ensure it continues to meet the changing needs and requirements of the organization. After considering all these factors, the appropriate approach can be chosen that best suits the processing needs and constraints. By taking a thoughtful and thorough approach to data ingestion, organizations can build an enterprise-level data pipeline that is scalable, efficient, and effective.

As a final note, it is imperative to highlight the difference between *data ingestion* and *data integration.* They are two distinct processes in the data pipeline, but they are closely related and often used together. Data ingestion is the process of acquiring data from various sources and loading it into a data storage system for further processing. It involves identifying the data sources and their formats, connecting to them, extracting the data, and transforming it into a format suitable for storage and processing. The focus is on moving the data from the source to the storage system in a timely and efficient manner. Data integration, on the other hand, is the process of combining data from multiple sources and presenting it as a unified view. It involves mapping and transforming data from different sources to a common format and schema and then loading it into a central repository or data warehouse. The focus is on making the data usable and accessible for analysis and decision-making. While data ingestion and data integration have different goals, they often go hand-in-hand. Data ingestion is the first step in the data pipeline, and it lays the foundation for data integration. Without reliable and consistent data ingestion, data integration becomes difficult or impossible. Conversely, data integration often requires data from multiple sources, and data ingestion is how that data is acquired. Several open-source tools can help with data ingestion and integration, each with its strengths and capabilities. Here are some examples. **Apache Kafka** is a distributed streaming platform that is designed to handle high volumes of data in real time. It can be used for data ingestion, data processing, and data integration across multiple systems and applications. Kafka supports message queuing and pub-sub messaging models and provides strong durability and fault-tolerance guarantees. Some advantages of Kafka include its scalability, high throughput, and flexibility in supporting a wide range of use cases. Many companies, such as LinkedIn, Netflix, and Airbnb, use Kafka for their real-time data needs. Moreover, **Apache Nifi** is a data integration and processing tool that is designed to automate data flows between systems and applications. It supports a variety of data sources and destinations and provides a graphical interface for building and managing data flows. Nifi supports data transformation, routing, and enrichment, and includes built-in security and governance features. Some advantages of Nifi include its ease of use, extensibility, and support for real-time and batch data processing. Companies such as NASA, Cisco, and Symantec use Nifi for their data integration needs. One last software which is worth noting is **Apache Airflow**. It is a workflow automation tool that is designed to manage and orchestrate data pipelines. It supports a variety of data sources and processing frameworks and provides a powerful workflow definition and scheduling engine. Airflow supports data validation, transformation, and quality checks, and includes built-in monitoring and alerting capabilities. Some advantages of Airflow include its flexibility, extensibility, and ease of deployment. Companies such as Airbnb, Twitter, and Lyft use Airflow for their data pipeline orchestration needs. In summary, data ingestion is focused on acquiring and loading data into a storage system, while data integration is focused on combining and transforming data from multiple sources into a unified view. Both are important steps in the data pipeline and are often used together to achieve the ultimate goal of making data usable for analysis and decision-making.

2.5 Data Storage

Data storage is a crucial stage in the data pipeline for machine learning models, involving the storage and management of data in a persistent manner. This process includes selecting appropriate storage technologies, designing data storage schemas, defining data storage requirements, and implementing data storage mechanisms for efficient and secure data storage. From a technical perspective, data storage involves using databases or file systems, such as relational or NoSQL databases and traditional or distributed file systems. Data storage is important as it enables efficient data management, ensuring data integrity, security, and scalability. By focusing on data storage, a robust system can be built, facilitating reproducibility, and effective data management practices. Challenges in data storage include managing large volumes of data, ensuring data consistency and accuracy, handling data versioning and lineage, and dealing with data security and access control. To avoid scalability issues, selecting scalable storage technologies, designing data storage schemas that can accommodate data growth, and leveraging distributed file systems or cloud-based storage solutions are key considerations. Optimizing and automating the data storage step can be achieved through built-in optimizations in data storage solutions, implementing automated data storage workflows using tools or frameworks, and leveraging data lake architectures or data warehouses for centralized and optimized data storage. Overall, optimizing and automating data storage can streamline the data pipeline, reduce manual efforts, and enhance the performance and scalability of machine learning models. Furthermore, in today's data-driven world, organizations are faced with the challenge of managing and analyzing vast amounts of data from various sources, including cloud-based data architectures. These architectures often result in data silos, where applications, users, and data are isolated, leading to inconsistencies and inaccuracies in data analysis. As a result, organizations are increasingly seeking innovative solutions to unify data, AI, and analytic workloads in a single source of truth. In this paper, we will explore three emerging solutions for data architecture, namely **data warehouse**, **data lake**, and **data marts**, and their potential to address these challenges in more detail.

2.5.1 Data Warehouse and Marts

The utilization of complex cloud data architectures has become routine for many organizations in their day-to-day business operations. However, these architectures often result in siloed applications, users, and data, leading to challenges in obtaining a single source of truth for data analytics and working with outdated data. To address these challenges, innovative solutions are emerging as the new standard for data architecture, with the goal of unifying data, artificial intelligence (AI), and analytic workloads in one centralized location. One such solution is the data warehouse, which is a type of data management system specifically designed to facilitate and support business intelligence (BI) activities, particularly analytics. Data warehouses are typically focused on performing queries and analysis and often store large amounts of historical data. The data within a data warehouse is usually derived from diverse sources such as application log files and transaction applications, which are consolidated into a centralized repository. The analytical capabilities of a data warehouse allow organizations to extract valuable business insights from their data, enabling improved decision-making. Over time, data warehouses build a historical record of data, which can be invaluable for data scientists and business analysts. There are different data warehouse architectures available, each with its advantages and disadvantages. Architectures such as centralized data warehouse, independent/dependent data mart, and hetero/homogeneous distributed data warehouse. However, in this section, overall differences and similarities between data warehouses and data marts are discussed, and how they can aid in establishing highly scalable pipelines [29]. As defined by Oracle, this centralization of data makes the data warehouse an organization's "single source of truth," providing a reliable and consistent foundation for data analysis and reporting [28]. A typical data warehouse may include several elements, such as a relational database for storing and managing data, an ELT solution for preparing data for analysis, statistical analysis, reporting, and data mining capabilities, client analysis tools for visualizing and presenting data to business users, and other advanced analytical applications that leverage data science and AI algorithms

or features for graph and spatial analysis at scale. The architecture of a data warehouse may vary depending on the specific needs of an organization. Common architectures include a simple design where metadata, summary data, and raw data are stored within the central repository of the warehouse, and data is fed from data sources and accessed by end users for analysis and reporting. Another approach is to add a staging area for data before it enters the warehouse, to simplify data preparation. Additionally, some organizations may implement a hub and spoke architecture, where data marts are added between the central repository and end users to customize the data warehouse for different lines of business. Sandboxes are another approach, providing private and secure areas for informal data exploration without conforming to formal data warehouse rules and protocols. Data warehouses have evolved over time since their inception in the late 1980s, initially serving the purpose of facilitating data flow from operational systems to **decision-support systems** (DSSs) [28]. Early data warehouses required significant redundancy as multiple DSS environments were often used to serve various users within organizations. However, as data warehouses became more efficient, they transformed into broader analytics infrastructures supporting a wide range of applications, including operational analytics and performance management. Data warehouse iterations have continued to progress over time, leading to the development of **enterprise data warehouses** (EDW) that deliver incremental value to organizations. A newer approach is the use of cloud data warehouses, which leverage the capabilities of cloud computing to ingest and store data from disparate sources. While traditional on-premises data warehouses still offer advantages such as improved governance, security, and data sovereignty, cloud data warehouses provide benefits such as elastic and scale-out support for large or variable compute or storage requirements, ease of use and management, and cost savings. The best cloud data warehouses are *fully managed* and *self-driving*, making it easy even for beginners to create and utilize a data warehouse with just a few clicks. Organizations can also choose to run their cloud data warehouse on-premises, behind their data center firewall, to comply with data sovereignty and security requirements.

In recent years, organizations have been actively seeking effective ways to make better and faster decisions about their businesses. One area where new ideas and concepts are gaining traction and proving to be useful from a corporate perspective is data warehousing. Data warehousing encompasses a set of techniques and technologies that enable organizations to integrate, store, and analyze data from various computerized systems used within the organization. With features such as data mining and ad hoc querying, data warehousing has become a crucial solution for organizations to gain insights and make informed decisions based on their data. Subsequently, the centralized data warehouse architecture, which involves creating a single, enterprise-wide information repository where all the organization's data is homogenized and integrated into a unified structure controlled by a single database product located in one location, was one of the earliest approaches to data warehousing [28, 29]. However, this architecture has faced challenges in meeting evolving business needs and providing decision-makers with instant access to critical data. High risk, long implementation timelines, high costs, and management difficulties are some of the drawbacks associated with the centralized data warehouse architecture [28, 29]. As a result, organizations started exploring other data warehouse architectures, such as **data marts**.

Data marts are smaller data warehouses that are built for specific departments or business units within an organization. They are usually created in isolation and have their own data models, metadata, and business information. On the other hand, there is another type of it called dependent data marts, which are built on top of a centralized data warehouse and depend on it for data integration and management. While these architectures offer some advantages in terms of localized data management and faster development cycles, they may still suffer from issues related to data consistency, integration, and scalability. To overcome the limitations of centralized data warehouses and independent/dependent data mart architectures, distributed data warehouse architectures have gained popularity in recent years. Distributed data warehouses are composed of multiple data marts that are distributed across different locations and interconnected through communication networks. These architectures offer greater flexibility, scalability, and cost-effectiveness compared to centralized data warehouses. They can

quickly adapt to changing business needs and accommodate the integration of new data marts without disrupting the entire system. However, there are challenges associated with distributed data warehouses, particularly when dealing with independent and separate data marts that have been created in advance. These data marts are often heterogeneous in nature, with different data models, metadata, and business information. This poses a significant challenge for developers of distributed data warehouses to reconcile and integrate these diverse data sources, resulting in increased complexity and potential data quality issues.

Moreover, in the current corporate landscape, data warehousing is gaining increasing traction as a valuable tool for managing and analyzing data across multiple computerized systems in organizations. Data warehousing allows for the creation of a centralized repository of homogenized and integrated data, providing a comprehensive view of an organization's activities and enabling information sharing among various departments. The concept of data warehousing is straightforward, but the initial centralized data warehouse architecture has faced challenges in providing decision-makers with instant access to critical data, such as high risk, long implementation timelines, high costs, and management difficulties. As a result, the need for adaptable and flexible data warehouse architectures has become paramount in meeting evolving business needs and providing real-time information to decision-makers. An optimal data warehouse architecture proposed by Kashfi et. al, which is discussed in the Literature Review section, builds upon the strengths of distributed data warehouses, including flexibility, scalability, and cost-effectiveness. It takes into consideration the specific needs and requirements of the organization and its environment, aiming to address the limitations of previous designs and achieve a desired solution for effective data warehousing. By leveraging the distributed nature of data warehousing and addressing the challenges associated with integrating independent data marts, the proposed architecture aims to provide a tailored solution for organizations to make better and faster decisions about their businesses. It is important to note that the optimal architecture may not be suitable for every situation or project, and customization for specific environments is necessary. In conclusion, the importance of data warehousing as a solution for integrating and analyzing data from various computerized systems in organizations cannot be understated. However, the traditional centralized data warehouse architecture has faced limitations in providing instant access to critical data, and the need for more adaptable and flexible architectures has emerged. The proposed optimal architecture aims to overcome the drawbacks of previous designs and provide a tailored solution for organizations to make better and faster decisions about their businesses, taking into consideration the specific needs and requirements of the organization and its environment. Further research and customization for specific environments are necessary to implement the proposed architecture effectively.

2.5.2 Data Lake

Data lakes are modern data storage solutions that are designed to handle large volumes of diverse data types, including structured, semi-structured, and unstructured data in their native format [28, 30]. Data lakes provide a flexible and scalable approach to data storage, allowing organizations to ingest, store, and analyze data in its raw, original format without the need for predefined data schemas or data transformations. There are two main types of data lakes: on-premises data lakes and cloud-based data lakes. On-premises data lakes are deployed within an organization's own data center or infrastructure, utilizing technologies such as Hadoop or Apache Spark to store and process data [28]. On the other hand, cloud-based data lakes are hosted on cloud platforms such as Amazon S3, Google Cloud Storage, or Microsoft Azure Blob Storage, leveraging cloud computing resources for data storage and processing [29]. Data lakes are employed for several reasons. One of the main advantages of data lakes is their ability to store and process large volumes of data of various types, making them suitable for organizations that deal with big data or have diverse data sources. Data lakes also offer flexibility in data exploration and analysis, as data can be stored in its raw format and transformed or analyzed as needed, allowing for agile and iterative data processing. Additionally, data lakes can integrate with other data processing technologies and tools, such as data pipelines, machine learning frameworks, and data

visualization tools, making them a versatile solution for modern data management needs. However, there are also some disadvantages to using data lakes. One challenge is the lack of predefined data schemas, which can make it more difficult to ensure data quality and consistency. Moreover, data lakes also require robust data governance practices to manage data security, access control, and compliance with data protection regulations. Additionally, data lakes can require specialized skills and expertise in data lake technologies and data management practices, which may pose a learning curve for organizations that are new to these technologies. Nevertheless, data lakes can deal with various types of data, including structured data (such as traditional databases, and spreadsheets), semi-structured data (such as JSON, and XML), and unstructured data (such as text, images, and videos) which makes data lakes suitable for handling diverse data sources, such as customer data, sensor data, social media data, and log data. In the context of machine learning, data lakes play a crucial role in improving the scalability of the data pipeline. Data lakes allow organizations to store and process large volumes of raw data in a flexible and scalable manner, enabling efficient data ingestion, data exploration, and data preparation for machine learning models. By storing data in its raw format, data lakes provide the flexibility to adapt to changing data requirements and data processing needs, which is essential for scaling up machine learning pipelines. Data lakes also enable organizations to leverage distributed processing capabilities, such as Apache Spark or Hadoop, to handle large-scale data processing tasks, making them ideal for machine learning workloads that require processing large volumes of data in parallel.

2.5.3 Requirements of Data Storages

When it comes to building a scalable data lake, organizations need to adhere to best practices, techniques, and methods to ensure effective data management. One crucial aspect is data governance, which involves implementing robust practices to ensure *data quality*, *consistency*, *security*, and *compliance with data protection regulations*, as discussed in Subchapter 2.1.1.2. This includes defining data access controls, establishing data lineage, and monitoring data quality to ensure that the data stored in the data lake is reliable and accurate.

Another critical consideration is data ingestion, as efficient and scalable data ingestion processes are essential for handling diverse data sources and formats, including batch processing and real-time data streaming. Organizations need to design ingestion processes that can handle data from various sources, such as databases, data streams, and external data sources, and efficiently load it into the data lake for further processing and analysis.

The data lake architecture is another crucial aspect to consider when building a scalable data lake. Organizations need to design a data lake architecture that is flexible and scalable enough to handle large volumes of data and accommodate changing data requirements. This includes considering factors such as data lake topology, data partitioning, and data storage formats. For instance, adopting a distributed file system, such as *Apache Hadoop HDFS* or *cloud-based object storage*, can enable scalable and distributed data storage, which is crucial for handling big data. Furthermore, leveraging data lake technologies that support data partitionings, such as *Apache Parquet* or *Apache ORC*, can optimize data retrieval and processing performance by organizing data in a columnar format and compressing it for efficient storage and query processing.

Finally, metadata management, that is, establishing a robust metadata management strategy to enable *data discovery*, *data lineage*, and *data cataloging*. This includes documenting metadata, such as data schema, data source, data quality, and data transformation history, to provide context and understanding of the data. The nuance of the topic is further explored in the next paragraph, shedding light on additional aspects and considerations that warrant attention.

2.5.4 Data Lake Topology

Data Lakes can be conceptualized as comprehensive repositories for data. However, there is flexibility in dividing them into distinct layers, typically ranging from 3 to 5 layers, which can be applied to most use cases. These layers, namely Raw, Standardized, Cleansed, Application, and Sandbox serve specific

purposes in the Data Lake architecture [30]. The **Raw** data layer, also known as the *Ingestion Layer* or *Landing Area* [30], serves as the initial repository for ingesting data into the data lake. The primary objective of this layer is to ingest data swiftly and efficiently in its native format, without any transformations. Data is preserved in its original state, allowing for the retrieval of historical data and preventing any overrides. However, even in the Raw layer, data organization is crucial, and it is recommended to categorize data into folders based on subject area, data source, object, and ingestion date. Access to the Raw layer should be restricted to end users, as the data in this layer is not readily consumable and requires in-depth knowledge for appropriate consumption. The Raw layer is akin to the staging area in traditional data warehousing. The **Standardized** data layer, though considered optional in most implementations, can be a beneficial addition if the data lake is expected to grow rapidly [30]. The main objective of this layer is to enhance performance in data transfer from the Raw to the Curated layer. This layer encompasses both daily transformations and on-demand loads. Unlike the Raw layer where data is stored in its native format, in the Standardized layer, data is transformed into a format that is optimal for cleansing. The structure of this layer is similar to the Raw layer, but it may be partitioned to a lower granularity if required. The **Cleansed** data layer, also referred to as the *Curated* or *Conformed layer* [30], is where data is transformed into consumable data sets and stored in files or tables. By this stage, the purpose of the data and its structure are well-known. Cleansing and transformations are performed before data is stored in this layer, and denormalization and consolidation of different objects are common practices. Organizing data in this layer follows a simple and straightforward structure, such as Purpose/Type/Files [30]. Typically, end users are granted access only to this layer, as it contains data that is ready for consumption. Furthermore, the **Sandbox** data layer is another optional layer that caters to advanced analysts and data scientists. This layer provides a space for conducting experiments to identify patterns or correlations in the data. It can also serve as the designated area for enriching data with external sources from the internet. Finally, the **Application** data layer, also known as the *Trusted*, *Secure*, or *Production layer*, is sourced from the Cleansed layer and enforced with any necessary business logic [30]. This layer may include surrogate keys shared among applications, row-level security, or other application-specific requirements. If machine learning models are utilized in any of the applications consuming data from the Data Lake, they are typically sourced from this layer. The structure of the data remains the same as in the Cleansed layer.

To facilitate a thorough comprehension of the logical topology of the distinct layers within a data lake architecture, Figure 6 serves as a visual representation of these concepts:

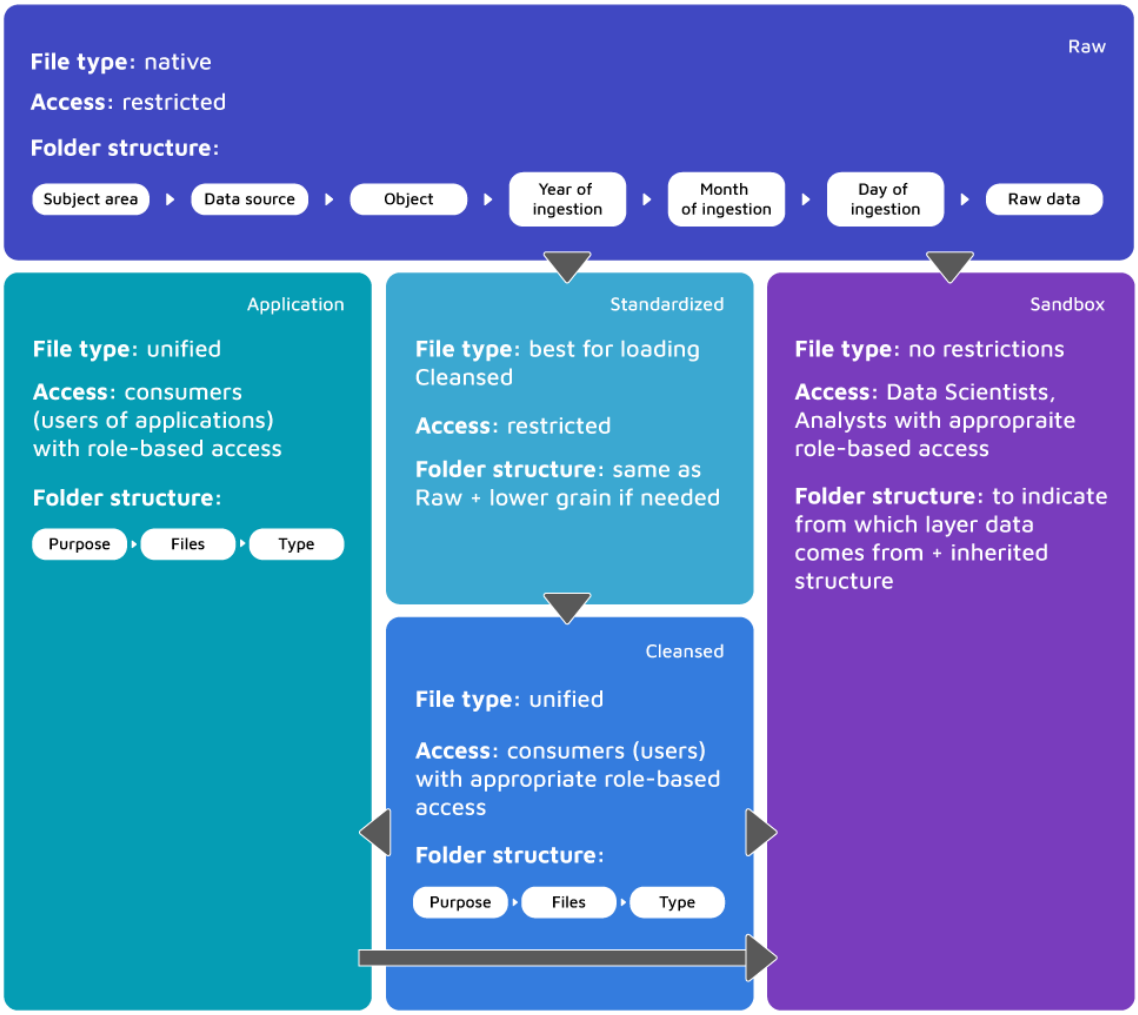

Figure 6: Logical Topology of the Data Lake Architecture [30].

In summary, the Data Lake architecture can be divided into distinct layers, each serving a specific purpose in the data processing and consumption journey. While the *Raw*, *Cleansed*, and *Application* layers are typically essential, the *Standardized* and *Sandbox* layers may be optional depending on the requirements of the Data Lake implementation. Proper organization, data transformations, and access control in each layer are crucial factors to consider when building a scalable and efficient data lake solution.

2.5.5 File Formats, Compression Algorithms & Encoding Types

Data formats and compression algorithms are critical components in optimizing data pipelines for scalability by mitigating the challenges associated with the size and complexity of the data being processed. Through the strategic utilization of suitable data formats and compression algorithms, organizations can effectively reduce the volume of data being processed, leading to improved efficiency and resource utilization in data pipeline operations. The choice of data format and compression algorithm significantly impacts the scalability of data pipelines, as it directly influences the processing time and resource requirements. Various data formats and compression algorithms can be employed to optimize data pipelines. For instance, columnar data formats such as Apache Parquet store data in a column-wise manner, as opposed to row-wise storage. This columnar approach allows for more efficient compression and storage, as well as faster querying and processing, ultimately enhancing scalability. Similarly, binary data formats like Apache Avro store data in a compact binary format, which is more efficient compared

to text-based formats, reducing the overall storage requirements and processing time. Furthermore, compression algorithms, such as gzip and Snappy, can be employed to reduce the size of data before it is stored or processed. These algorithms employ different compression techniques, such as lossless compression, to effectively reduce the volume of data, leading to reduced storage requirements and improved processing efficiency. The use of compression algorithms in data pipelines can significantly enhance scalability by reducing the amount of data that needs to be transferred and processed, resulting in faster processing times and reduced resource utilization. By strategically selecting the appropriate data format and compression algorithm, organizations can optimize their data pipelines for scalability. This optimization can lead to reduced processing time, improved performance, and enhanced capacity to handle larger volumes of data more efficiently. The careful consideration and implementation of data formats and compression algorithms in data pipelines are crucial for organizations seeking to optimize their data processing operations and achieve scalable and efficient data pipeline architectures.

In the realm of big data analytics, the selection of appropriate data serialization and storage formats is crucial for optimizing data processing performance. Two critical factors to consider are the volume of data and the type of data processing tasks, such as Online Transaction Processing (OLTP) and Online Analytical Processing (OLAP). This paragraph introduces the three popular data formats, namely **ORC** (Optimized Row Columnar), **Avro** (Apache Avro), and **Parquet** (Apache Parquet), which are known for their efficiency in handling high volumes of data, particularly OLTP and OLAP data. ORC (Optimized Row Columnar) is a columnar storage format specifically designed for big data analytics. It is optimized for use with Apache Hive and Apache Pig, making it well-suited for OLAP data processing tasks. ORC utilizes columnar storage to minimize the amount of data that needs to be read, resulting in faster processing of large data sets. Additionally, ORC employs compression algorithms such as Snappy and Zlib to reduce the size of stored data, further improving performance. Avro (Apache Avro) is a data serialization format that offers compact binary encoding and schema evolution capabilities. It supports dynamic data structures and enables data to be easily shared and processed across multiple systems. While Avro is not as optimized for scalability as ORC and Parquet, it offers flexibility in handling various data types and structures, making it suitable for both OLTP and OLAP data. Avro allows for customization of the compression algorithm used, including options such as Snappy and Gzip, based on the specific requirements of the data and processing tasks. Parquet (Apache Parquet) is a columnar storage format that is widely recognized for its optimization in big data analytics. It supports a variety of compression algorithms, including Snappy and Gzip, and provides options for encoding techniques such as bit-packing, run-length encoding, and delta encoding. Parquet is highly efficient in reducing the size of stored data, which leads to faster reading and processing of large data sets. Furthermore, Parquet is designed to be compatible with a wide range of data processing systems, making it a suitable choice for handling OLTP and OLAP data across multiple systems. In summary, ORC, Avro, and Parquet are three popular data formats that are well-suited for efficient data processing in the context of high volumes of OLTP and OLAP data. ORC is optimized for use with Hive and Pig, while Avro offers flexibility in handling various data types and structures. Parquet is widely recognized for its optimization in big data analytics and compatibility with various data processing systems. The selection of an appropriate data format among these options depends on specific requirements and considerations, such as the type of data and processing tasks involved in a given big data analytics use case.

Data compression algorithms are commonly used to enhance the scalability of machine learning pipelines by reducing the size of data that needs to be stored and transmitted. This can lead to reduced storage and processing requirements, enabling the pipeline to handle larger datasets and improve overall performance. Several data compression algorithms are known for their efficiency in handling high volumes of data. These algorithms include **Snappy**, **Zlib**, **LZO**, **Gzip**, **LZ4**, and **Deflate**. The characteristics of compression algorithms can have a significant impact on the scalability and efficiency of machine learning pipelines when dealing with big data. In this section, we will discuss three key characteristics of compression algorithms: compression ratio, decompression speed, and memory usage, and how they can affect the performance of a machine learning pipeline.

Compression Ratio: Compression ratio refers to the reduction in the size of the data after it has been compressed. A higher compression ratio means that the data takes up less space, which can be beneficial for reducing storage requirements and for faster transmission of data over networks. The compression ratio of different algorithms can vary significantly. Gzip provides the highest compression ratio, followed by Zlib and Deflate, while Snappy and LZO have lower compression ratios but faster decompression speeds. The choice of compression algorithm will depend on the specific requirements of the application, including the amount of data to be processed and the desired compression ratio.
Decompression Speed: Decompression speed refers to the speed at which the compressed data can be decompressed and restored to its original form. A fast decompression speed is crucial in big data and machine learning pipelines because it can reduce the time it takes to process the data, which in turn can improve the overall performance of the pipeline. Different compression algorithms have varying decompression speeds. Snappy and LZ4 are known for their fast decompression speeds, followed by LZO, Zlib, Deflate, and Gzip. The choice of compression algorithm will depend on the specific requirements of the application, particularly the need for quick decompression in processing large amounts of data.
Memory Usage: Memory usage refers to the amount of memory required to perform the compression or decompression operation. In big data and machine learning pipelines, memory usage is a critical consideration as it can significantly impact the performance of the pipeline. Compression algorithms with low memory usage can help reduce the amount of memory required to process the data, thereby improving the performance of the pipeline. Snappy and LZ4 are known to have lower memory usage compared to the other algorithms, while Gzip has the highest memory usage. The choice of compression algorithm will depend on the available memory resources and the specific requirements of the application.

Selecting the most suitable compression algorithm for handling big data and scaling infrastructure depends on the specific requirements of the application. In general, Snappy and LZ4 are suitable choices for applications that require fast decompression speeds, such as in big data processing pipelines where large amounts of data need to be decompressed quickly. On the other hand, Gzip is a good choice for applications that require high compression ratios, such as in data archiving and backup systems. Zlib and Deflate are suitable for applications that require a balance between compression ratio and decompression speed. There are opportunities for further improvement of compression algorithms by enhancing their compression ratio, reducing decompression time, or minimizing memory usage. This can be achieved through the development of new algorithms that employ advanced compression techniques or leverage hardware acceleration for better compression ratios or faster decompression speeds. Additionally, existing algorithms can be optimized for specific use cases, such as big data processing or data transmission over networks. The implementation and pseudocode for these algorithms may vary depending on the programming language and library used. However, the basic steps for compression involve reading the input data, compressing the data using the specific algorithm, and writing the compressed data to an output file. For decompression, the basic steps include reading the compressed data, decompressing the data using the specific algorithm, and writing the decompressed data to an output file. In conclusion, data compression algorithms play a vital role in improving the efficiency and scalability of machine learning pipelines for handling high volumes of OLTP and OLAP data. The selection of the most suitable algorithm depends on the specific requirements of the application, considering factors such as compression ratio, decompression speed, and memory usage. Further research and optimization of compression algorithms can contribute to enhancing their performance in handling big data and scaling infrastructure.

In conclusion, data lakes are powerful data storage solutions that can significantly improve the scalability of the data pipeline for machine learning. By providing flexible and scalable data storage and processing capabilities, data lakes enable organizations to efficiently handle large volumes of diverse data, prepare data for machine learning, and leverage distributed processing for scalable machine learning workloads. Following best practices, techniques, and methods for building a scalable data lake,

such as data governance, data ingestion, data transformation, data lake architecture, metadata management, data security and compliance, monitoring and performance optimization, and data lake backup and disaster recovery, can help organizations build a robust and scalable data lake that supports their machine learning initiatives.

2.6 Data Versioning

Data versioning is a crucial step in the data preparation stage of the data pipeline for machine learning models. It involves managing different versions of data used in the machine learning process to ensure consistency, traceability, and reproducibility of results. In technical terms, data versioning typically involves creating a version history or snapshot of the data at different points in time. This can be achieved through various methods such as creating copies of data files or database tables, using version control systems (e.g., Git), or leveraging data versioning tools (e.g., DVC or Data Version Control). These versions can be labeled, tagged, or timestamped, allowing for easy identification and retrieval of specific data versions. Data versioning is essential for several reasons. Firstly, it enables data scientists and machine learning practitioners to track changes in data over time, ensuring that the same data is used consistently throughout the machine learning process. This helps in maintaining the reproducibility and traceability of results, which is critical for model evaluation, validation, and deployment. Secondly, data versioning allows for the rollback or recovery of previous data versions in case of errors or issues, providing a safety net for data integrity. Lastly, data versioning supports collaborative data science workflows, where multiple team members may be working on the same dataset simultaneously, enabling effective collaboration and data sharing. However, there are challenges associated with data versioning, such as data storage requirements, data consistency, and data synchronization. Managing multiple versions of data can increase storage overhead, especially for large datasets, and may require efficient storage solutions such as distributed file systems or cloud storage. Ensuring data consistency and synchronization across different versions can be complex, particularly when dealing with updates, deletions, or modifications to data. Additionally, versioning can also introduce complexities in managing dependencies, metadata, and access controls, which need to be carefully addressed. To avoid scalability issues, there are key points to consider when implementing data versioning. Firstly, data versioning should be planned and documented as part of the overall data pipeline architecture, with clear guidelines and standards for versioning practices. Secondly, data versioning should be automated as much as possible to minimize human error and ensure consistency. This can be achieved through the use of versioning tools, scripts, or workflows that can automatically capture changes in data and create version snapshots. Additionally, optimization techniques such as data differencing, data compression, or delta encoding can be applied to reduce storage overhead and improve efficiency.

2.6.1 Data Versioning Approaches

Versioning data has always been valuable, but its significance has increased in modern data environments where data is used to support mission-critical business processes. As organizations embrace cloud technologies, managed services, and engineering practices, data versioning is becoming an essential aspect of managing the complexity of data ecosystems. In this paper, we will explore three instructive approaches to implementing data versioning, starting from basic forms and moving toward more advanced solutions. These approaches include full duplication, "valid_from/to" metadata and first-class data version control [31]. The first approach, **full duplication**, will be examined as the starting point. Full duplication is a method of versioning data where a complete copy of the dataset is saved in a new location each time a version is created. This means that every object in the dataset, whether it has changed or not, is duplicated in each version. This approach can be used in various use cases, such as tracking changes in a small dataset over time, creating daily or periodic snapshots of data, or maintaining historical records of data changes. One advantage of full duplication is its simplicity and ease of implementation. It can be easily implemented using a standard file system or database operations to create new copies of the dataset in separate locations. This makes it a straightforward approach for data

versioning, especially for smaller datasets and when the versioning frequency is low. However, one of the limitations of full duplication is its scalability. As the size of the dataset grows or the frequency of versioning increases, the storage requirements also grow proportionally. This can result in significant storage redundancy, as unchanged data objects are duplicated in each version, leading to increased storage costs and inefficiencies. In summary, a full duplication is a simple approach for versioning data, suitable for smaller datasets with infrequent versioning needs. It can be easily implemented using a standard file system or database operations. However, it may not be the most scalable solution for larger datasets or frequent versioning requirements, as it can result in storage redundancy and increased costs. Careful consideration of the specific use case and dataset requirements is necessary when implementing full duplication for data versioning. A more space-efficient and incremental approach, which involves the utilization of **"valid_from/to" metadata**, is considered next. The use of "valid_from/to" metadata is a method of versioning data where changes to the dataset are recorded using metadata indicating the validity period of each version. Instead of duplicating the entire dataset for each version, only the changes made within a specific period are recorded, along with the valid_from and valid_to timestamps indicating the period during which the changes are valid. The "valid_from/to" metadata approach is commonly used in scenarios where changes to the dataset are frequent, and it is important to maintain a historical record of data changes without duplicating the entire dataset for each version. This can be useful in scenarios such as tracking changes to a continuously updated dataset, maintaining audit trails of data modifications, or enabling temporal querying of data. One of the advantages of using "valid_from/to" metadata is its efficiency in terms of storage space. Only the changes made within a specific period need to be recorded, reducing the storage requirements compared to full duplication. Additionally, code or queries that interact with the versioned data can reference the valid_from and valid_to timestamps, making it easier to retrieve the correct version of the data based on a specific timestamp or time range. Implementing "valid_from/to" metadata typically involves adding additional metadata fields, such as valid_from and valid_to timestamps, to the dataset or database schema. These timestamps can be automatically updated whenever changes are made to the data, and queries or codes that interact with the data can be modified to consider the validity period of each version. In short, the use of "valid_from/to" metadata is an efficient approach for versioning data, particularly in scenarios where changes to the dataset are frequent. It allows for maintaining a historical record of data changes without duplicating the entire dataset for each version and can be implemented by adding additional metadata fields and updating queries or code to consider the validity period of each version. Consideration of the specific use case and dataset requirements is important when implementing "valid_from/to" metadata for data versioning. Finally, the concept of first-class data version control, whereby versioning is treated as an inherent property of data, is explored next. **First-class version control** is a method of versioning data that treats data versioning as a core feature of the data management system, rather than an external process. In this approach, changes to the dataset are tracked and managed within the data management system itself, similar to how version control systems like Git are used for source code management. One of the main use cases for first-class version control is in collaborative data management scenarios, where multiple users or teams need to work on the same dataset simultaneously, and changes need to be tracked and merged in a controlled manner. This can be particularly useful in data science projects, where different team members may be working on different aspects of the data and need to merge their changes into a single coherent version. First-class version control can also be helpful in scenarios where data changes need to be audited, reviewed, or rolled back. It allows for fine-grained control over data versions, enabling data managers to track changes, review modifications, and revert to previous versions if needed. In terms of scalability, first-class version control can help manage large datasets efficiently by tracking only the changes made to the data, rather than duplicating the entire dataset for each version. This can reduce storage requirements and enable more efficient management of data versions over time. Implementing first-class version control typically involves using data management systems or tools that provide built-in versioning capabilities. These systems may offer features such as version tracking, merging, branching, and rolling back, similar to the

functionalities provided by traditional version control systems for source code management. Users can interact with the data and its versions through a version control interface, allowing them to manage changes, track modifications, and collaborate on the data. In summary, first-class version control is a data management approach that treats data versioning as a core feature of the system, and it can be particularly useful in collaborative data management scenarios. It provides benefits such as efficient storage, fine-grained version control, and the ability to track, review, and roll back data changes. Implementation of first-class version control typically involves using data management systems or tools with built-in versioning capabilities.

3 LITERATURE REVIEW

Over the years, extensive research has been conducted on the topic of data engineering, with the significance and relevance of this field of study recognized by scholars and practitioners alike. In this section, a comprehensive overview of the existing body of knowledge on the existing or still developing solutions is provided, including their historical background, theoretical frameworks, and key concepts. By gaining an understanding of the literature on these concepts, the significance of data engineering can be better appreciated, and research gaps that need to be addressed in the study can be identified. The topic of data preparation in today's data-driven landscape is of utmost significance and relevance to the field of study. With the exponential growth of data volumes and complexity, enterprises face significant challenges in managing and processing data for meaningful insights and informed decision-making. Data preparation, the process of converting raw data into a usable format, plays a pivotal role in the data pipeline and has a direct impact on downstream data analysis and modeling. The historical background of data preparation can be traced back to traditional methods that have been widely used in enterprises. However, these traditional methods may struggle to keep up with data's increasing volumes and complexity, leading to scalability issues, inefficiencies, delays, and suboptimal performance in the data pipeline. This creates a pressing need to explore and optimize data preparation methods to address these challenges. Theoretical frameworks and cutting-edge tools and methodologies, such as Apache Nifi, Apache Atlas, and Apache Spark, are extensively employed. These frameworks and technologies offer potential solutions for enhancing the scalability of data preparation in enterprise-grade data pipelines. The literature review draws insights from industry practices, state-of-the-art technologies, and existing literature to propose practical strategies and recommendations for designing a scalable data strategy in an enterprise setting. The significance of this study lies in its actionable insights and recommendations to enhance the performance of data pipelines in enterprise-grade data environments. The proposed strategies and recommendations are intended to address the limitations and bottlenecks identified in existing data preparation methods, tools, and technologies, and provide a roadmap for designing a well-optimized data preparation pipeline that incorporates scalable data ingestion, efficient data transformation, and intelligent data storage strategies. The study emphasizes the need for robust data preparation processes in enterprises dealing with large volumes of data to ensure reliable and efficient data processing in the data pipeline. The subsequent sections of this chapter aim to expound upon the main concepts, theories, and models that are pertinent to the topic at hand. Furthermore, a comprehensive review of the existing literature will be conducted to discern the key theories or frameworks that have been employed in prior research related to the topic and elucidate their relevance to the current study.

Data Governance

In recent years, the complexity of data governance has increased due to the growing number of stakeholders and activities associated with big data. Accessing and managing data has become challenging due to its distribution across various organizations and heterogeneous sources. The variety of sources across different administrative environments further complicates data management, implementing data governance frameworks a difficult task. The first paper to be reviewed is titled "An Ontological-Based Model to Data Governance for Big Data" authored by Alfonso Castro and his colleagues [5]. This paper proposes one potential solution to address this challenge, which is, the use of

ontologies. Ontologies offer significant advantages in knowledge representation, including their formalization and expressiveness, which can be applied to various fields, including data governance [5]. In this paper, the authors propose an ontology that covers the entire vision of the data governance framework, encompassing all data processes and their relationships. The objective of data governance is to ensure that all data management actions are carried out accurately. The inference process generates new knowledge about the operation of data management actions and controls their correct performance, triggering corrective actions in case of any problems. For instance, in the case of security management, if an alarm indicates that a field in a file that should be anonymized is not, the inference process generates signals that initiate processes to anonymize these fields. The core element of the proposed architecture is the **z** (SKP), which governs the entire data management process [5]. It includes the *semantic model*, *instances*, and *reasoning* rules that govern the behavior of the autonomous components. The ontology-based description of the domain and management rules within the SKP reduces the complexity of the components. The SKP is composed of the combined knowledge planes from each component, and the modification of one component's part can affect the behavior of other components. The first step in building the SKP is designing a conceptual model that describes the domain and its associated ontology. OWL (Web Ontology Language) has been chosen as the implementation language, offering advantages such as the ability to use reasoning tools in artificial intelligence [5]. The second step is implementing the rules that govern the system behavior using SWRL (Semantic Web Rule Language), which extends the abstract syntax of OWL to include conditional rules in the ontology language. The proposed model is built upon an autonomous system comprising distributed components that employ semantic techniques and automatic ontology-based reasoning. The system includes a Shared Knowledge Plane through which the components interact, with a core ontology representing the various data management processes involved in data governance. The authors have implemented and tested a prototype of the proposed model for Telefonica's global video service, and the results obtained indicate the feasibility of using this approach to reduce the complexity of managing big data environments.

The paper presented herein is deemed to be of particular significance in the realm of data governance for the proposed data pipeline, as it lays the foundation for the establishment of a robust data governance system. Based on the information provided, a data governance system for enterprises with multiple data teams and data sources can be constructed through the following steps [5]:

1. **Data Process Identification**: The data processes entailed in data governance for Company A are to be identified based on the provided ontology. These processes encompass data security, data quality, data architecture management, data modeling and design, data storage and operations, metadata management, data warehousing, and business intelligence, as well as data integration and interoperability.
2. **Activities and Roles Definition**: For each data process, the activities associated with it, such as planning, control, development, and operational activities, are to be defined. Additionally, the roles of the various actors involved in data management, including supplier roles, responsible roles, stakeholder roles, and consumer roles, are to be identified.
3. **Classes and Subclasses Definition**: Classes and subclasses are to be created in the ontology to represent the distinct entities and activities that are involved in data governance. For instance, classes for Entity, Activity, Agent, Data Process, and Event can be created, with subclasses delineating specific activities within each data process.
4. **Attributes Definition**: Attributes for each class and subclass are to be defined to capture relevant information, such as type, name, generation date, access permission, cipher status, start date, object entity, type of activity, and completion status.
5. **Relationships Definition**: Relationships between the classes and subclasses in the ontology are to be defined to capture the interconnections between different entities and activities that are part of data governance. For instance, relationships between Entity and Activity can be defined to represent the activities performed on entities, and between Agent and Activity to denote the responsibility of agents for activities[5].

6. **Description Language Utilization**: A suitable description language, such as OWL or SWRL, is to be chosen to represent the ontology and capture the rules and relationships between properties [5].
7. **Implementation of the Data Governance System**: The data governance system is to be implemented utilizing the defined ontology and description language. This may involve creating a data governance framework, defining processes and procedures for data management, assigning roles and responsibilities to data teams, establishing data quality standards, implementing data security measures, setting up data warehousing and business intelligence systems, and integrating data from different sources.
8. **Monitoring and Evaluation**: The effectiveness of the data governance system is to be continuously monitored and evaluated to ensure that it is meeting the data management needs of Company A. Adjustments and improvements are to be made as necessary based on feedback and performance metrics.
9. **Ontology Update**: The ontology is to be updated as necessary to reflect changes in data governance processes, activities, roles, or relationships over time.

By adhering to these steps, a comprehensive data governance system can be established for guaranteeing the security, quality, and appropriate utilization of data throughout the organization, and facilitating effective data management and decision-making.

Another article discusses the importance of data governance in today's organizations [32]. Data governance refers to the set of policies, procedures, and practices that ensure the effective and efficient use of data in an organization. It involves managing the availability, usability, integrity, and security of data used in an organization to help achieve business goals [32]. The article identifies six key attributes that enterprise data must have: accessibility, availability, quality, consistency, audibility, and security [32]. Data governance manages and develops these attributes to enhance the overall value of data as an asset to the organization. To ensure that enterprise data meets all these attributes, an effective data governance framework involves four key components: standards, policies and processes, organization, and technology [32]. The standards component establishes the standards for data in an enterprise, such as data definitions, taxonomies, and enterprise data models. The policies and processes component is responsible for creating, developing, controlling, managing, and auditing data [32]. The organization component assigns ownership of data, defines roles and responsibilities, and ensures that executives and staff from both the line of business and the IT department are involved. Finally, the technology component provides an infrastructure that automates and scales the development and enforcement of data governance standards, policies, and processes. The article then delves deeper into two of the four data governance components, namely organization, and technology [32]. A clear definition of roles and responsibilities is critical for any governance program, including data governance. Companies need to be rigorous about defining roles and assigning specific responsibilities to individuals involved in data governance to enforce accountability. A common factor among companies with successful data governance initiatives is that executives and staff from both the line of business and the IT department are involved. The business assumes ownership of the data and takes the lead in driving data governance, while IT partners with the business to implement the technical aspects of the data governance program. Technology is a crucial component of data governance, as it provides an infrastructure that automates and scales the development and enforcement of data governance standards, policies, and processes. A data integration technology platform that provides built-in capabilities to access, cleanse, transform, deliver, and monitor data is ideal for data governance. Just as a business process management solution helps to streamline business processes, a data integration platform helps to automate data-related processes. In conclusion, data governance is essential for organizations looking to effectively and efficiently use their data to achieve their business goals. By ensuring that enterprise data is accessible, available, of high quality, consistent, auditable, and secure, data governance enhances the overall value of data as an asset to the organization [32]. The four key components of data governance - standards,

policies and processes, organization, and technology - work together to establish a comprehensive framework for managing and developing data governance attributes [32]. Clear roles and responsibilities and business and IT involvement are critical organizational success factors, while technology provides an infrastructure that automates and scales the development and enforcement of data governance standards, policies, and processes [32].

Spark Lineage with Spline

The article discusses the increasing popularity of Apache Spark in the Hadoop ecosystem as a replacement for the initial computation engine, MapReduce [33]. This shift is mainly due to Spark's improved and simplified usability and significant performance benefits. Additionally, while many big data tools are being ported to Spark, the paper notes that Apache Atlas, a data governance and metadata framework for Hadoop, lacks support for Apache Spark. Furthermore, the Spark Web UI visualization is too low-level and hard to understand, especially for non-technical people. To address these issues, the authors have designed and implemented a proof-of-concept tool called Spline that captures lineages of Spark SQL computations in a fast, unobtrusive, and asynchronous way [33]. These lineages are exported into a database and visualized in a human-friendly manner. The paper describes the principles behind the proposed design, the implementation details, experimental results, future work, and a conclusion. In addition, the article provides an overview of the various components and levels of abstraction in Apache Spark, including RDDs, DataFrames, and the Catalyst optimizer, and how these are used in Spark SQL [33]. The authors note that the logical plans of Datasets used in Spark SQL computations can describe the full flow of data through the Hadoop environment and operations applied to it at the attribute level. This type of data lineage is valuable for financial institutions and other organizations attempting to comply with regulations. The Spline tool was developed to be lightweight, unobtrusive, and easy to use, and the article describes how these objectives were achieved in the implementation [33]. Moreover, the article discusses the implementation and testing of a data lineage tool called Spline for Apache Spark. The tool is designed to capture data lineage for regulatory compliance purposes in financial institutions using big data tools. The article begins by describing the solution design of Spline, which leverages Spark's optimization stages and the QueryExecutionListener to access execution plans containing data lineage information [33]. The tool is resilient and handles potential exceptions raised in user-defined handlers. Spline implements its listener to access QueryExecution instances and harvests lineage information, which is transformed into a Spline model and stored in Spline storage [33]. Users can browse harvested lineages through the Spline Web UI. By default, Spline persists the lineage model to its internal storage, specifically to a MongoDB database, but can also be configured to push lineage information to Apache Atlas or JSON files on a Hadoop file system. The article presents two experiments to test Spline's performance. The first experiment uses a simple Spark application that filters input data for records from 2016 and English articles [33]. This data is grouped by hour of the day and the number of page views is summed. The experiment is run on different data sizes (10mil. to 100mil. rows) with and without Spline, and the total processing time and time used by Spline are measured [33]. The results show that Spline's performance cost is low and constant for all runs, and it does not depend on the processed data size. The second experiment reads 100 mils. rows dataset and executes randomly generated transformations ranging from 100 to 1000 operations, producing logical plans of different lengths [33]. The processing time and memory consumption of Spline are measured. The results show that the processing time and memory consumption of Spline linearly grew with the growing number of operations in the logical plan, as expected. Overall, the article provides valuable insights into the increasing adoption of Apache Spark and the challenges faced by organizations in capturing data lineage for regulatory compliance purposes. The development and implementation of Spline as a lightweight, simple-to-use and fault-tolerant tool for capturing Spark data lineage is a significant contribution to the big data ecosystem. The experiments demonstrate that Spline's performance cost is low and constant for all runs, and its computational complexity is linearly dependent on the number of operations in the logical plan. Future work can focus on extending Spline's capabilities to capture the flow of data through

numerous Spark applications and the Hadoop cluster overall and visualizing how lineage changes over time.

Data scalability is a critical consideration in the design and implementation of machine learning systems. As data volumes increase, the processing and storage requirements can quickly become a bottleneck, limiting the ability of the system to handle large datasets and complex models. This can result in longer processing times, increased costs, and decreased performance, ultimately impacting the accuracy and reliability of the system. One of the main challenges of scaling data is ensuring that the system can handle large volumes of data without sacrificing performance or accuracy [19, 20]. This requires careful consideration of the data storage and processing architecture, including factors such as data partitioning, load balancing, and parallel processing [20]. Additionally, the system must be able to adapt to changing data volumes and requirements over time, without requiring significant modifications or downtime. Another challenge is ensuring that the system can handle a variety of data types and formats, including structured, semi-structured, and unstructured data [21]. This requires flexible data processing pipelines and the ability to handle a range of data inputs and sources, from databases and files to streaming data and IoT devices. To avoid these problems, it is essential to consider several key points. First, it is crucial to plan for scalability from the beginning of the project, rather than as an afterthought. This involves selecting the right *data storage* and *processing technologies*, designing the system *architecture for scalability*, and establishing clear goals and metrics. Second, it is important to prioritize data quality and data governance to ensure that the data used in the system is accurate, complete, and consistent. This can help avoid issues with *data inconsistency* or *duplication*, which can impact the accuracy and reliability of the system. Finally, it is essential to monitor and optimize the system performance over time, using metrics such as *data throughput*, *processing time*, and *accuracy*. This can help identify and address bottlenecks in the system, optimize resource allocation, and ensure that the system is meeting scalability goals and requirements.

There are several data integration and preprocessing tools available, ranging from open-source to commercial products. Popular open-source tools include Apache Spark, Apache Kafka, and Apache NiFi, which offer features such as *data ingestion*, *processing*, and *transformation*. These tools can also be integrated with other machine learning tools and frameworks, making them a flexible and powerful option. Moreover, commercial data integration and preprocessing tools, such as Informatica, Talend, and IBM InfoSphere, provide additional features such as *data quality*, *governance*, and *lineage*. They can be used to manage complex data pipelines and workflows, ensuring that data is consistent and reliable throughout the pipeline.

In the paper by Imawan et al. [26], two approaches for scalable extraction of timeline information from road traffic data were introduced: (1) an *iterative MapReduce approach* and (2) a *single MapReduce approach*. The iterative MapReduce approach utilizes multiple iterations of MapReduce, whereas the single approach involves only one iteration of MapReduce. First off, the authors described how they divided the iterative MapReduce approach into two parts: (1) *detecting congestion events* and (2) *updating the dependency of each event*. They explained the details of how to extract the timeline information using one machine in their previous work. In the paper, they propose MapReduce-based algorithms for scalable extraction [26]. The basic iterative MapReduce approach involves running two iterations of MapReduce jobs. In the first iteration, congestion events are detected from the given daily speed log data. The output size is much smaller than the input size after this step. The second iteration updates the dependency of each event generated from the first iteration. The final output is a set of congestion events without the inter-date events. To start, the mapper of the first iteration takes the raw traffic data for each date, which is represented as a list of records containing a date, time, road link ID, and speed value. The mapper produces a key-value pair consisting of a date and a road link ID as the key and a pair of time and speed as the value. This process is described in more detail in the paper [26]. In the basic version of this approach, the entire process of extracting timeline information is carried out at the mapper. The input of this mapper is the daily speed log data and its output is daily congestion events. These inputs and outputs are the same as those of the first mapper and the output of the second

reducer of the iterative approach. To illustrate an example of the single MapReduce approach for the basic application, the red box in Figure 7 is used. With the same input as the iterative approach, the mapper detects the congestion event and directly updates its dependency. As there is no additional task for the reducer, the mapper uses date D as the key, and there is no explicit reducer needed for the single approach. Similarly, the mapper of the single approach is explained in the paper [26].

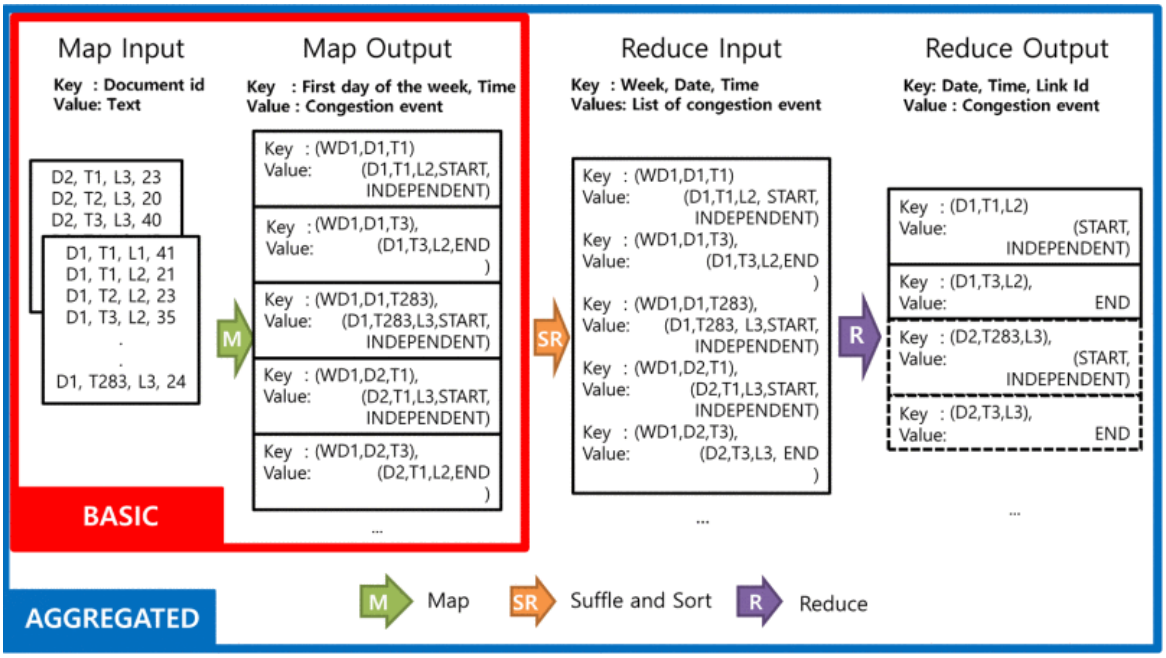

Figure 7: Single MapReduce Approach [26].

As a concluding remark, it is important to highlight that prioritizing this step is also important for building a scalable and flexible system that can handle changing data sources or requirements. Subsequently, this future-proofs the system and allows for easier maintenance and updates as needed. It is necessary to retrieve relevant data from various sources and format it appropriately for ingestion into an ML model. Inaccurate or biased results can occur if the data extraction process is not carefully considered and executed, which can ultimately impact the effectiveness of the model and lead to poor decision-making. By selecting the right data sources and extracting data properly, the risk of overfitting or underfitting the model can be minimized, leading to a better generalization of new data. In summary, carefully considering and executing the data extraction process is essential to building a robust machine learning system that produces accurate and reliable results, reduces the risk of costly mistakes, and future-proofs the system for scalability and flexibility.

Data Profiling

Another article by Aikoh, K et. al introduces a unique approach to metadata profiling that aims to enhance the quality of metadata management in data catalogs [34]. The current process of data preparation is laborious for data scientists, and the proposed method is expected to ease this burden while enabling enterprises to efficiently utilize existing data assets. Although data catalogs have gained traction as a solution, conventional data profiling methods do not verify metadata accuracy, making it challenging to identify erroneous metadata [34]. To address this issue, the authors' method emphasizes the structural similarity of meta-graphs that depict the relationships between data and metadata. This involves the generation of a meta-graph by a graph representation generator, the measurement of data source similarity using structural similarity by a similarity calculator, and the identification of incorrect metadata by a metadata profiler, which detects conflicts that arise when different metadata are assigned for similar data sources. The authors posit that similar data sources possess similar metadata as business data generally follows implicit rules such as business rules, industry common sense, system

specification, or the same producer. Additionally, the article stresses the role of the Chief Data Officer (CDO) in enhancing data utilization within a data management system. While data management can be complex without a well-established methodology, the CDO is tasked with reviewing governance policies constantly, ensuring conflict management is adhered to, and developing governance policies to mitigate risks associated with publishing the minimum necessary information [34]. The authors also note that short-term development and design based on implicit domain knowledge and assumptions can hinder data reuse [34]. IBM and Microsoft have proposed data management systems with a data catalog to establish the data governance method that eliminates the tradeoffs between analytics cost and operation cost [34]. These catalogs register metadata, including meaning, features, generation method, and management policy, for each data source. Nevertheless, metadata quality management is vital in the data catalog search system, and conventional data profiling methods do not sufficiently account for metadata, resulting in the detection of inaccurate metadata being challenging. The authors conducted experiments to demonstrate the efficacy of their proposed method. The results reveal that the method accurately detects erroneous metadata and reduces the time taken for data preparation significantly. The proposed method is projected to enhance data management systems by improving metadata quality and alleviating the workload of data scientists.

4 APPROACH & METHODOLOGY

4.1 Case Study

To illustrate the point, a real-life company will be presented as an example. However, in the interest of preserving confidentiality and protecting sensitive information, it will be necessary to anonymize or substitute certain details and datasets. By doing so, the identity and proprietary information of the company will be safeguarded while still providing a relevant and informative case study. Moreover, the techniques and methods discussed in this paper will be implemented, and several metrics used to evaluate the performance of a data pipeline will be compared. The following discourse aims to present a case study of a medium-sized retail company (say company A), which seeks to enhance its customer experience by offering personalized recommendations and augmenting the accuracy of its demand forecasting. In pursuit of this objective, the organization identifies two fundamental business capabilities that will play a pivotal role in its overarching goal, namely, (1) a Personalized recommendation engine and (2) Accurate demand forecasting. It should be noted that the company in question, Company A, has an existing data stack, hereafter referred to as Pipeline A, that is intended for similar purposes as the proposed pipeline, hereafter referred to as Pipeline B. As part of the evaluation process, Pipeline B is subjected to rigorous testing and benchmarking against Pipeline A to determine its effectiveness and efficiency in meeting the company's data needs.

The utilization of company data is expected to provide benefits to multiple teams, among which are the following teams that are anticipated to be significant beneficiaries:

Table 2: Mapping of the Data Domains

Team Name	Responsibility	Data Needs	Usage
Data Science Team	Conducting data analysis, modeling, and experimentation to generate insights and improve data products	Raw data, large amounts of data, historical data, real-time data, and metadata about the data	Modeling and Analysis
Marketing Team	Developing and executing marketing strategies, campaigns, and initiatives to promote products or services, attract and retain customers, and achieve business objectives	Customer data, user behavior data, market data, and competitive data	Conducting market research and analysis, analyzing marketing channel performance, developing targeted marketing campaigns, and measuring the success of marketing initiatives

Team Name	Responsibility	Data Needs	Usage
Sales Team	Gather and analyze data related to customer interactions and behavior	Customer behavior and preferences, demographics, and purchasing patterns, sales data, and market trends	Identify potential customers and clients, develop sales strategies
BI Team	Creating and managing business intelligence tools and dashboards that enable business stakeholders to access and visualize data	Customer data, sales data, financial data, and marketing data	Ensure that data is accessible, accurate, and relevant and that stakeholders can easily interpret and act upon the insights provided by the data

[a] This table provides a mapping of various data domains to the teams responsible for managing them, their corresponding data needs, and how the data is utilized by each team.

Table 2 provides a comprehensive overview of the data generated by the company, including the types of data, sources, and intended usage. This essential information lays the foundation for the subsequent steps involved in building a data pipeline. Specifically, the initial step of defining the data requirements has been accomplished through the identification of the specific data needed to achieve specific business objectives or resolve issues, as detailed in Chapter 2.1.1. This step ensures that the pipeline is tailored to the company's specific needs, avoiding unnecessary data collection and processing that can slow down the process and inflate costs. By providing a clear picture of the required data, Table 2 facilitates the design of a more efficient and effective data pipeline.

The subsequent step in the data pipeline development process involves the meticulous identification of data sources. In compliance with privacy regulations, a thorough examination of the data collection methods employed by the company will be elucidated. Furthermore, the nature of the data, including its characteristics, format, and structure, will be expounded upon, along with a comprehensive overview of the specific sources that are earmarked for ingestion into the pipeline. This step serves a crucial role in establishing a robust and secure data pipeline by carefully scrutinizing the origin, quality, and relevance of the data that will be fed into the pipeline. This meticulous approach ensures that only pertinent and authorized data is integrated into the pipeline, safeguarding against potential data breaches, inaccuracies, and inconsistencies that can undermine the integrity and reliability of the pipeline's outputs.

In this research study, the data sets and sources to be used for the experiment will be introduced. This step is crucial for designing an effective pipeline and conducting thorough data profiling. This section will provide an overview of the important metrics and characteristics that need to be considered during the collection process. However, for a more comprehensive and detailed description of each data set/source, reference to Appendix A will be made. The selection of appropriate sources is a critical step in any data-driven research or experiment. For this study, multiple data sets and sources have been carefully chosen to ensure the availability of relevant and reliable data for analysis. These practices are essential for constructing a robust data pipeline and conducting thorough data profiling, which are essential steps in the data analysis process.

When designing a pipeline and conducting data profiling, several important metrics and characteristics need to be considered. These metrics and characteristics play a crucial role in ensuring the quality and integrity of the data used for analysis. The collection of these metrics serves as a crucial step in establishing a robust foundation for effective governance, comprehensive profiling, and thorough lineage analysis. These metrics not only facilitate the accurate characterization and classification of data, but also provide invaluable insights into the quality, accuracy, and timeliness of the data. This, in turn, enables organizations to establish data management best practices and ensure the integrity and reliability of the data used for decision-making processes. Furthermore, the systematic collection and analysis of these metrics empower organizations to identify data quality issues, identify potential data lineage gaps, and establish data profiling protocols to mitigate risks associated with data inconsistency, inaccuracy, or incompleteness. Therefore, the meticulous collection of these metrics serves as a cornerstone in the development of a robust data governance framework, a comprehensive data profiling strategy, and a reliable data lineage analysis approach. In the data collection process, several key metrics and characteristics play a crucial role in ensuring the quality and reliability of the collected data. These

metrics provide insights into various aspects of the data, including its completeness, accuracy, consistency, validity, currency, and latency. To gain a comprehensive understanding of the key metrics and characteristics that are important during the data collection process, it is essential to delve deeper into the details of these terms. A thorough examination of these metrics can provide valuable insights into the quality and reliability of the collected data, which in turn can significantly impact the outcomes and conclusions of a research study. For further elucidation on these terms, Chapter 2.3 of the research report serves as a detailed source of information. This chapter delves into a detailed discussion of each of these metrics, providing a comprehensive overview of their significance in the context of data collection. It sheds light on the nuances, implications, and practical considerations associated with these metrics, providing a robust foundation for understanding their role in ensuring the quality of the collected data.

Furthermore, Appendix A of the research report serves as a valuable resource for gaining insights into the characteristics of each dataset and data source used in the research. This appendix provides detailed information on various aspects of the datasets, including their quality, collection method, schema (i.e., entries of the dataset), and what each entry represents. This information can be crucial in understanding the context and reliability of the data used in the research, as it provides a comprehensive overview of the data sources, their characteristics, and their relevance to the research study. The quality of the data used in a research study is of utmost importance, as it directly impacts the validity and reliability of the research findings. Appendix A provides a detailed assessment of the quality of each dataset, including measures such as data accuracy, completeness, consistency, and validity. This assessment provides a comprehensive understanding of the data quality and its implications for the research study, allowing researchers to make informed decisions and interpretations based on the quality of the data. In addition to the quality of the data, the collection method of the data is another critical aspect that can significantly impact the reliability of the research findings. Appendix A provides in-depth information on the collection method of each dataset, including details on the data collection process, data sources, and data collection techniques employed. This information allows researchers to evaluate the rigor and reliability of the data collection process, and to make informed judgments about the potential biases, limitations, or strengths of the data. Furthermore, Appendix A also provides insights into the schema of each dataset, which includes the entries or fields of the dataset and what they represent. Understanding the schema of the dataset is crucial for interpreting the data correctly and extracting meaningful insights. The schema information provided in Appendix A enables researchers to understand the structure and organization of the data and to identify the specific data elements that are relevant to their research objectives. In summary, the selection of appropriate data sets and data sources, as well as consideration of important metrics and characteristics during the data collection process, is crucial for ensuring the quality and integrity of the data used for analysis. The data capture frequency, data completeness, data accuracy, and data latency are important aspects to be considered in the data collection process, as they directly impact the reliability and validity of the analysis results. By carefully considering these metrics and characteristics, a robust data pipeline can be designed, and thorough data profiling can be conducted, laying the foundation for accurate and insightful data analysis in this research study. For a more detailed description of each data set/source, a reference to Appendix A is recommended.

Once clear boundaries of data domains have been defined, along with rules for data profiling and ownership, it becomes feasible to establish robust governance rules. Firstly, Corporation's data governance team collaborates with representatives from each of the four teams to define data governance policies. These policies cover areas such as data quality, data security, data privacy, data retention, and data sharing, and are aligned with the company's overall business objectives and strategies.

Table 3: Data Ownership Roles

Data Source	Team Name	Data Owner	Data Steward	Data Custodian
Transaction Data	Sales Team	STL	SDS	SDC

Data Source	Team Name	Data Owner	Data Steward	Data Custodian
Website Analytics	BI Team	BITL	BIDS	BIDC
Metadata and Logs	Data Science Team	DSTL	DSDS	DSDC
Social Media Data	Marketing Team	MTL	MDS	MDC
Product Review Data	Marketing Team	MTL	MDS	MDC
Sales Data	Sales Team	STL	SDS	SDC
Market Data	Market Data Team	MTL	MDS	MDC

[a] data ownership matrix, where the different teams within Company A are listed along with the data sources they deal with. For each data source, the matrix identifies the roles of the data owner, data steward, and data custodian. Explanation of Abbreviations: Sales (S), Marketing (M), Data Science (DS), BI Data (BI) plus Team Lead (TL), Data Steward (DS), Data Custodian (DC)

As depicted in Table 3, the "Data Owner" refers to the team or individual who is responsible for the overall management and governance of the data source, including setting data policies, defining data quality standards, and ensuring compliance with data regulations. The "Data Steward" is the person who is responsible for day-to-day data management tasks, such as data integration, data validation, and data cleansing. The "Data Custodian" is the person or team who physically manages and stores the data, ensuring data security and access controls. In this case, as mentioned in the prompt, logs are handled exclusively by data engineers, who are responsible for managing and processing the log's data source. There are either assigned to someone on this team, or to someone who is a new hire specifically for it.

Once data requirements have been defined and data sources have been established, including ownership and governance considerations, the next step is to identify appropriate methods for data extraction. This crucial phase involves selecting the most suitable techniques or tools for retrieving data from the identified sources. Several factors need to be taken into consideration, such as the type and format of data, volume of data, data source availability, data retrieval frequency, and scalability requirements. As demonstrated in Figure 8, the aforementioned concept is depicted in the flow diagram, which will further be built:

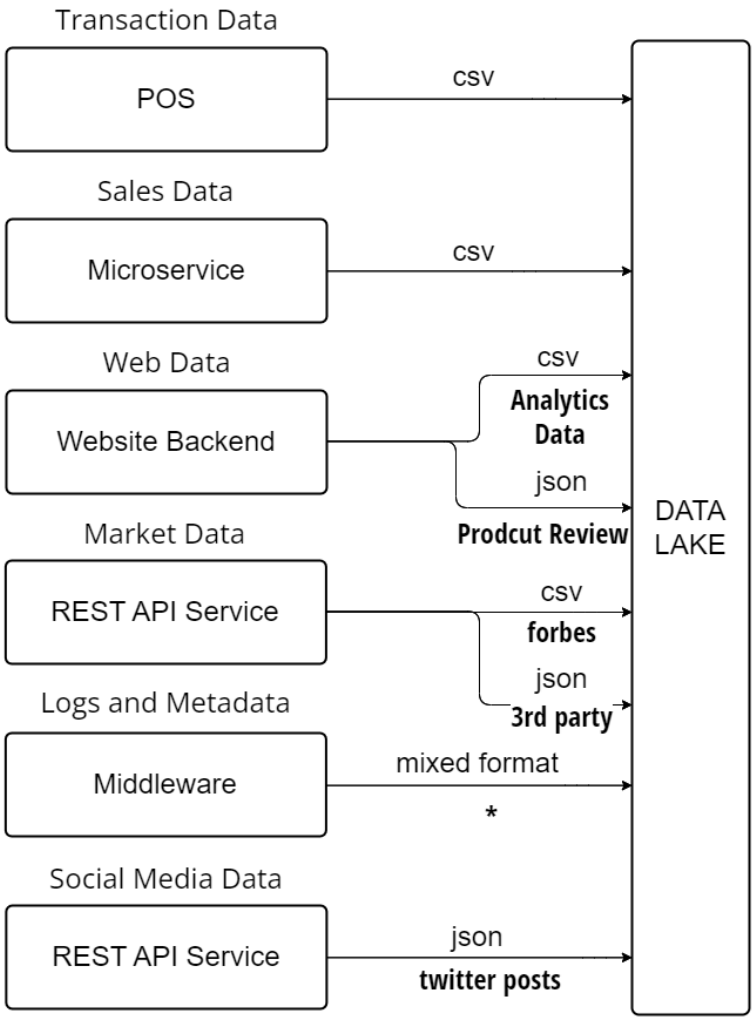

Figure 8: Flow Diagram depicting 6 data sources company A consumes, as well as, the format each dataset is being produced.

Moving forward, the subsequent step in the progression toward constructing a scalable data pipeline involves the establishment of a data lake. Based on the provided information on the six data sources (transaction data, sales data, web data, market data, logs & metadata, and social media data), a suitable data lake topology can be constructed. Data Lakes are typically organized into distinct layers, ranging from 3 to 5 layers, which can be adapted to various use cases. These layers, namely Raw, Standardized, Cleansed, Application, and Sandbox serve specific purposes within the Data Lake architecture [30]. Based on these layers, the data sources can be placed as follows; Initially, all of the data can be ingested into the raw data layer in its native format. Based on an analysis of the inherent characteristics of each data source and their intended use in the context of this case study, a recommended data lake architecture is presented in Figure 9. The initial step involves ingesting all data sources into the raw data layer in their original format. Subsequently, transaction data is directed to the standardized layer to establish a uniform schema, followed by the cleansed layer for preprocessing and data cleaning. Similarly, sales data, social media data, and market data are also routed to the standardized layer for similar purposes. Lastly, web data is directly processed in the application layer, where business logic is enforced and the data is utilized in applications, including machine learning models if applicable. This architecture is designed to ensure efficient data processing and utilization while maintaining data quality and

consistency throughout the data lake environment, aligning with best practices in data lake architecture. See Figure 9:

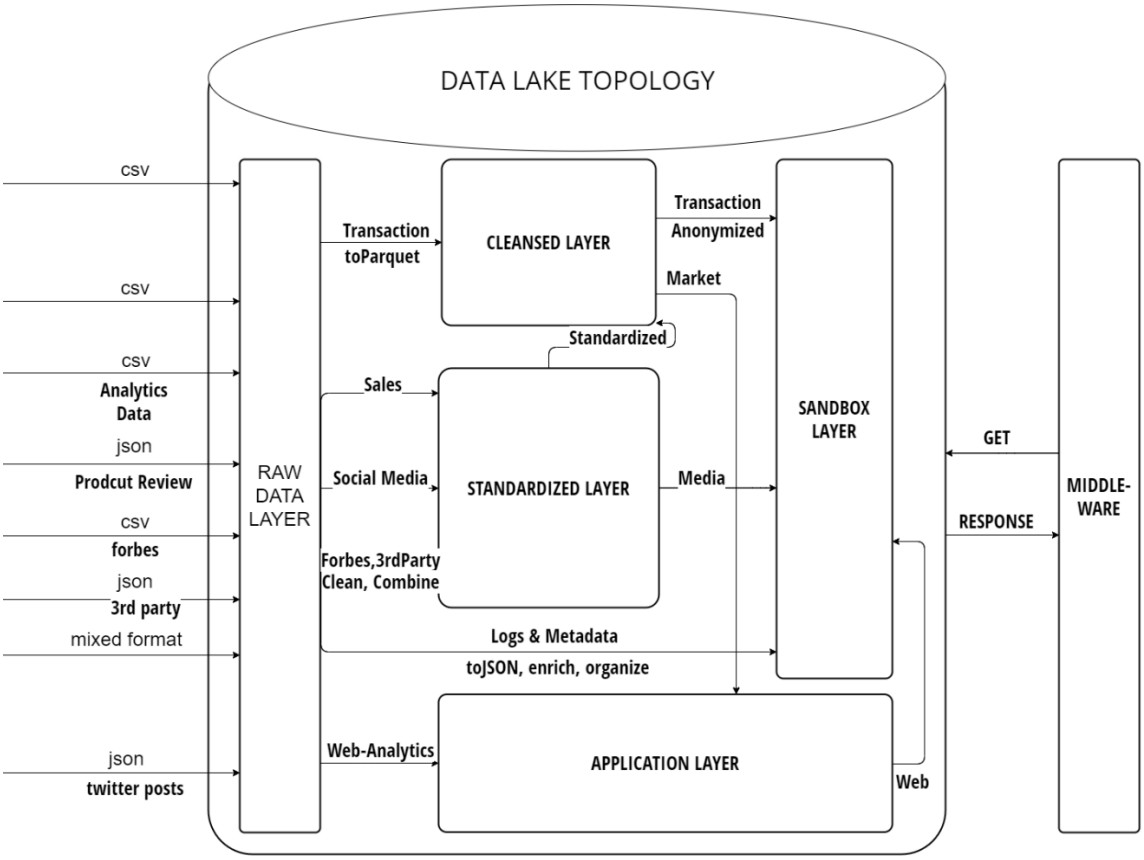

Figure 9: Illustration of the recommended data lake architecture where raw data from various sources is initially ingested into a raw data layer, followed by data standardization and cleansing in the standardized layer, and direct processing in the application layer for web data, to enforce business logic and enable utilization in applications.

Next up on the list is a Streaming platform that will allow the building of real-time data streaming applications and the processing of large volumes of data in a distributed and fault-tolerant manner. Apache Kafka, a distributed and open-source streaming platform, is an ideal choice for that. In the context of the data lake architecture illustrated in Figure 9, the previously depicted middleware is now replaced by the Kafka Service, which facilitates direct access to data from the data lake. However, there appears to be a need for a data ingestion service, which is addressed by leveraging Kafka Connectors. These Kafka Connectors serve as middleware for "publishing" and "subscribing" to Kafka streams, enabling seamless data ingestion and processing within the data lake ecosystem. One important feature provided by this middleware is the incorporation of **Nginx**, which facilitates load balancing and distribution of incoming data across multiple backend services. This capability is made possible through the implementation of the Data Mesh architecture, which enables efficient handling of large volumes of data and seamless integration of various data sources within the data lake ecosystem. This software bridge, referred to as the middleware, can be observed in Figure 10, depicting its role as an intermediate layer that facilitates the integration and communication between different components of the data lake architecture. In contrast,

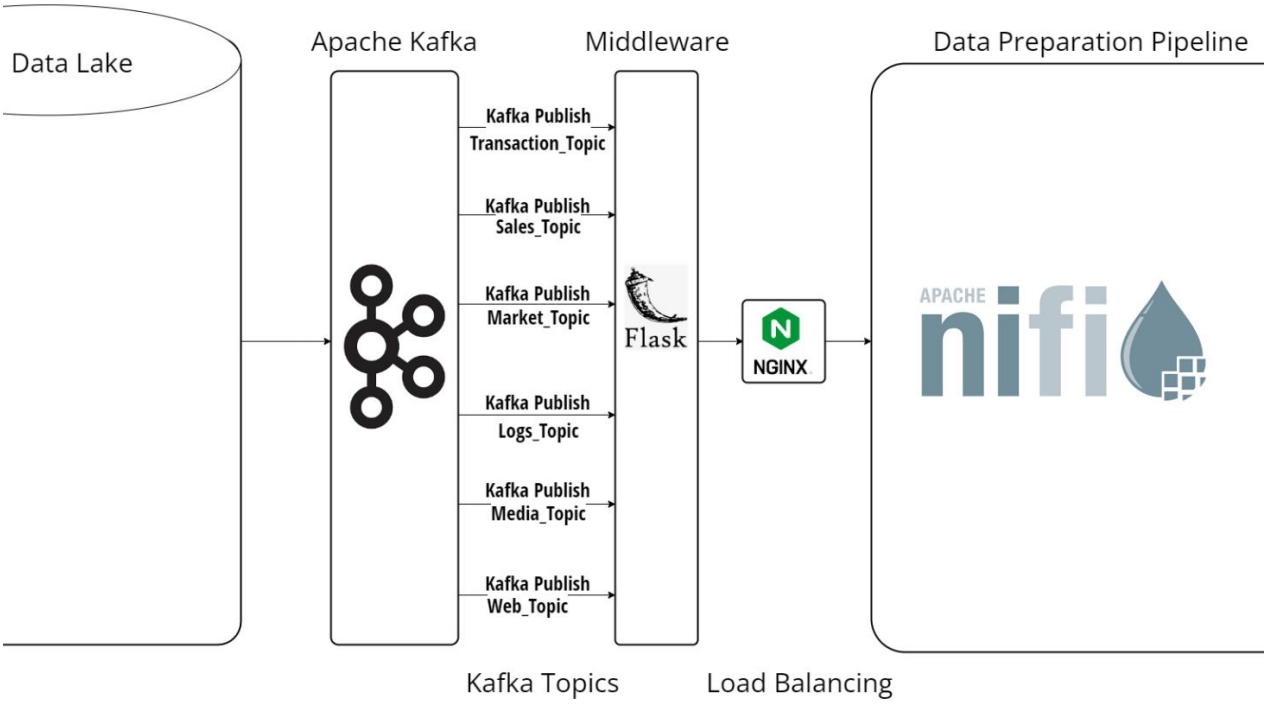

Figure 10: Data Meshed Architecture. Illustration of the connection between the data lake and Apache Kafka, where a middleware layer with Nginx acts as a bridge, referred to as Kafka Connectors, facilitating communication between Kafka topics and Apache NiFi flow.

In contrast, as depicted in Figure 11, the original pipeline design exhibited multiple flows, including excessive backpressure on the ETL process, leading to system stalls and failed processing cases due to the overwhelming load on the system:

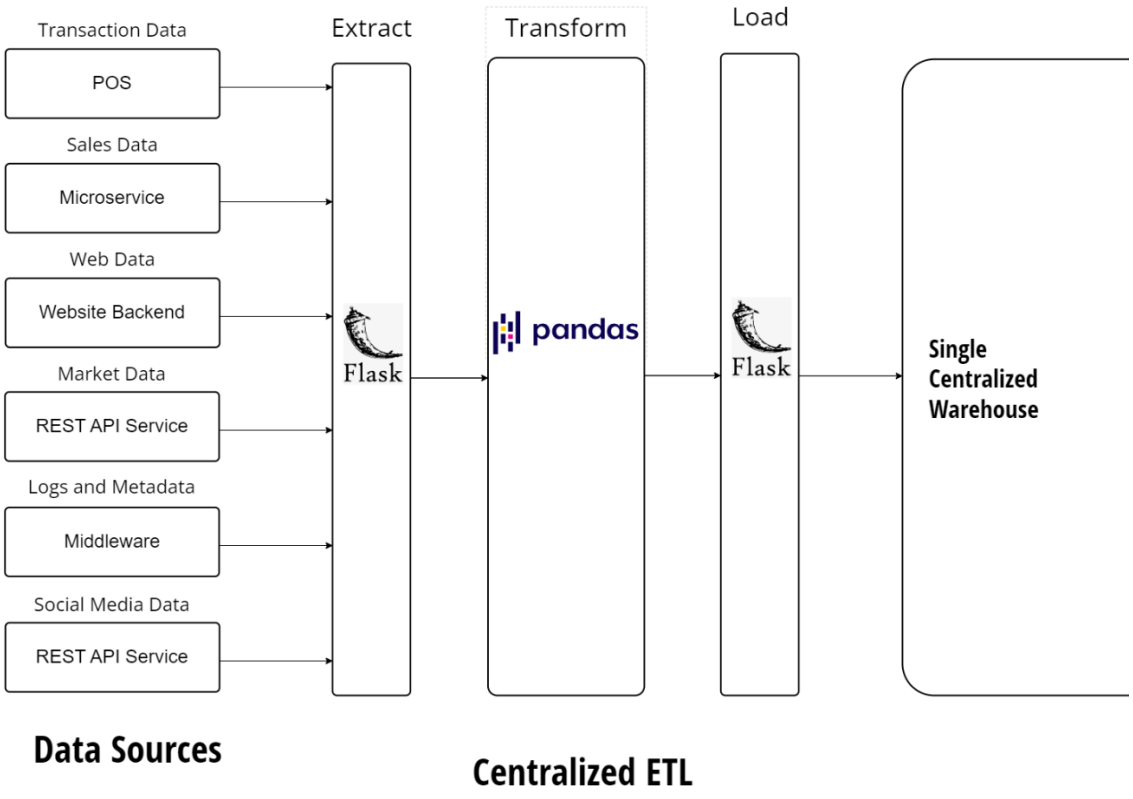

Figure 11: The original pipeline design that suffered from high backpressure on ETL, system stalls, and failed processing cases.

The subsequent step entails constructing the processing segment of the pipeline utilizing Apache Nifi. The proposed design will consist of two versions of the pipeline, one adhering to the original design with a predefined set of preprocessing steps, and the other incorporating the proposed solution with the same preprocessing steps. These two designs will be utilized in experiments to assess the throughput of the pipeline in terms of flow files, with each flow file containing a single batch of data. For instance, if the connector is feeding the pipeline with 5000 rows of batch per cycle, then each flow file will contain approximately 5000 rows and occupy approximately 0.12 MB of space. Furthermore, Figure 12 illustrates the data meshed pipeline design, showcasing the ingestion of data from Kafka connectors and the integration of Apache Spark processors into the Nifi pipeline through Apache Livy's HTTP bridge. Notably, the design incorporates flow file exchanges to reduce access time. In contrast to the traditional ETL pipeline that transforms and loads data into a centralized warehouse, this architecture eliminates certain access time delays, thereby enhancing performance and enabling more dynamic in-memory processing.

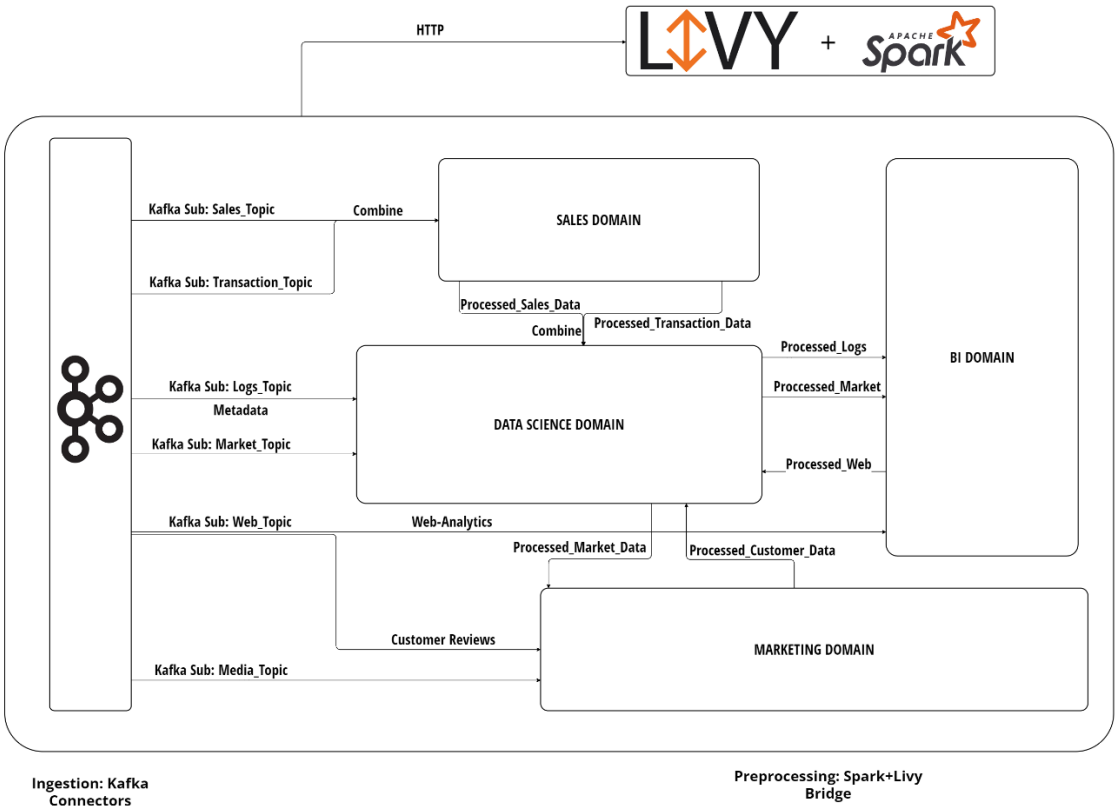

Figure 12: The data meshed pipeline design, showcasing the ingestion of data from Kafka connectors and the integration of Apache Spark processors into the Nifi pipeline through Apache Livy's HTTP bridge. Notably, the design incorporates flow file exchanges to reduce access time. In contrast to the traditional ETL pipeline that transforms and loads data into a centralized warehouse, this architecture eliminates certain access time delays, thereby enhancing performance and enabling more dynamic in-memory processing.

Following the design of an ETL plus reverse ETL approach, the data integration process begins with data extraction from various sources, followed by data transformation and storage in a data lake. Subsequently, the transformed data is loaded back into a data warehouse for further utilization. Upon completion of this process, each data team, such as the data science team for machine learning and predictive analytics, the BI team for report generation and visualization, the Sales team for sales analysis and forecasting, and the Marketing team for customer segmentation and campaign targeting, leverages the final product according to their specific needs. While how data is utilized may vary across different domains, a common element among all of them is the loading of data into the data warehouse. In the context of the current system, the adoption of a dependent data mart infrastructure appears to be the most appropriate approach. This is because Company A consists of four distinct teams, and to ensure effective data governance, each team must have its own encapsulated access to the data store. As a result, the proposed solution entails the implementation of four dependent data marts, each connected to a single centralized data warehouse. Among these data marts, the Data Science and Business Intelligence teams are allocated two marts each, and one data mart per team is shared with the Sales and Marketing teams. The design of this infrastructure is illustrated in Figure 13. The rationale behind this decision is that the Business Intelligence (BI) and Data Science (DS) teams are characterized as heavy processing teams, and ensuring the availability of critical data is of paramount importance for their operations. Hence, dedicated data marts are designated for storing critical data, while shared data marts are utilized for storing other types of data. This approach is designed to optimize data storage and accessibility for the specific needs of BI and DS teams, as depicted in Figure 13. Refer to Appendix B, Figure 14 - 17 to see the implementation in Apache Nifi.

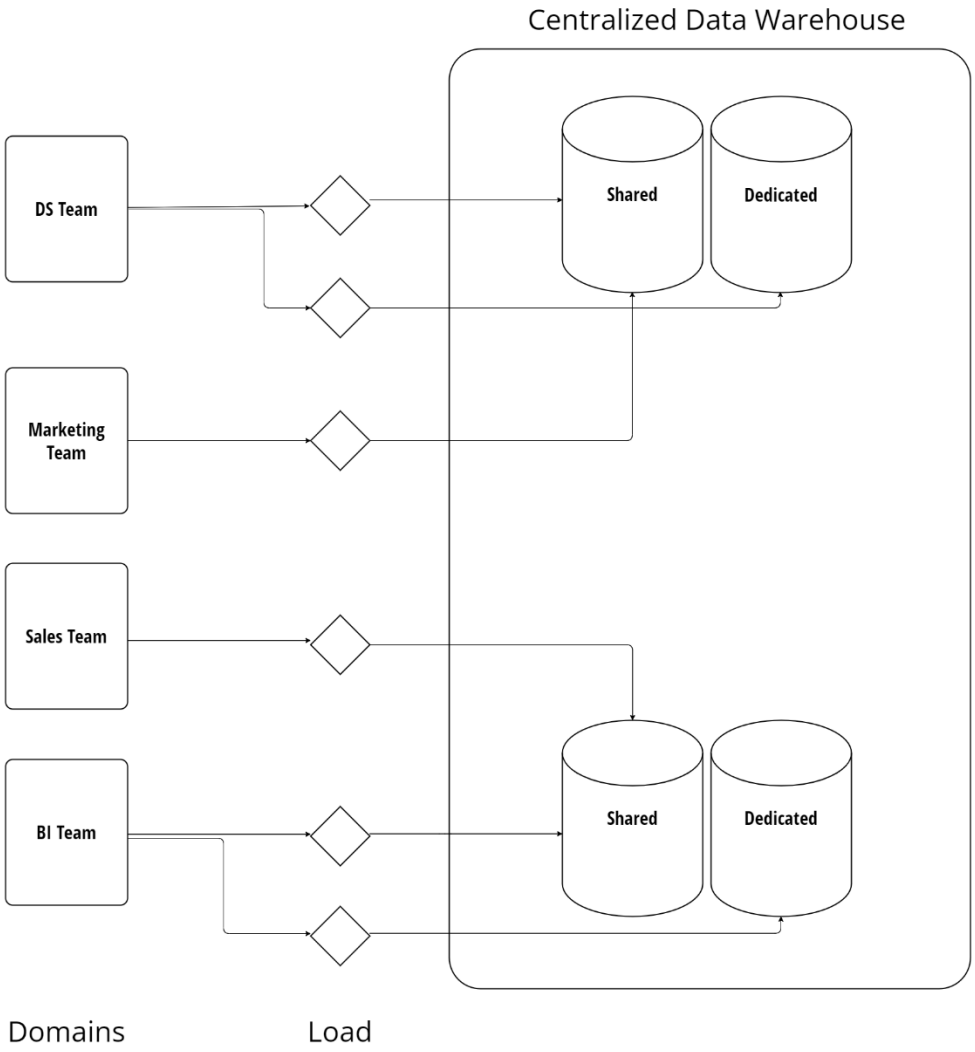

Figure 13: The proposed data mart infrastructure design, with 4 dependent data marts connected to a single data warehouse, where two dedicated data marts are allocated for the Data Science and Business Intelligence teams, while the remaining data marts are shared among the other teams.

4.2 Technical Details & Tools

4.2.1 Apache Atlas

Additional measures for data governance can be achieved through the utilization of Apache Atlas, an open-source data catalog solution. Apache Atlas provides a comprehensive set of features that facilitate data management and governance within an organization. With Apache Atlas, organizations can establish standardized processes for defining data sources, data lineage, data glossary, and data quality, fostering a consistent and organized approach to data governance. One of the key benefits of Apache Atlas is its ability to automate data discovery and metadata management. Through its data profiling capabilities, Apache Atlas can automatically analyze and assess data assets, providing insights into their quality, accuracy, completeness, and consistency. This enables organizations to identify data quality

issues and take corrective actions, ensuring that data assets meet the defined data governance policies and standards. Furthermore, Apache Atlas provides data lineage visualization, which allows users to visually trace the flow of data from its source to its destination, providing a clear understanding of data origins, transformations, and usage. This helps organizations establish data lineage and track changes and transformations applied to data along the data pipeline, ensuring data traceability and accountability. Additionally, Apache Atlas offers a data glossary feature that provides a common vocabulary and definition of terms used in the organization's data assets. This fosters consistency and understanding of data terminologies, minimizing data ambiguity and misinterpretation. Moreover, Apache Atlas supports data collaboration features that enable data users to collaborate and share comments, annotations, and feedback on data assets. This promotes collaboration among data users, facilitating data understanding and knowledge sharing across teams. In conclusion, Apache Atlas serves as a valuable tool for implementing data governance in organizations, providing automated data discovery, metadata management, data lineage visualization, data glossary, and data collaboration capabilities. By leveraging Apache Atlas, organizations can establish standardized processes, foster a consistent approach to data governance, and ensure data quality, traceability, and accountability throughout the data pipeline.

4.2.2 Apache Nifi

Apache NiFi is an open-source data integration platform that provides powerful capabilities for data flow control, orchestration, and management in a scalable and efficient manner. It offers a web-based graphical interface that allows users to design, configure, monitor, and manage data flows through a visual flow-based approach, making it suitable for academic and professional use. One of the main advantages of Apache NiFi is its robust flow control and orchestration capabilities, which allow users to design complex data flows with ease using its intuitive visual interface. NiFi provides a wide range of processors that can be used to ingest, transform, route, monitor, and manage data, and these processors can be easily connected to create data pipelines. Additionally, NiFi offers advanced features such as backpressure handling, flow prioritization, and load balancing, which enable efficient data flow control and management. Transition words such as "additionally" can be used to improve the coherence and flow of the paragraph. NiFi is also highly scalable, supporting distributed processing and clustering, allowing users to handle large volumes of data efficiently. This makes it suitable for handling data processing requirements in big data and high-velocity data environments. Transition words such as "in addition" can be used to connect the advantages of scalability with the previous point about flow control and orchestration. Furthermore, NiFi provides robust data lineage and provenance capabilities, which allow users to track the origin and movement of data throughout the data flow pipeline. This enables users to trace the flow of data, understand data transformations, and perform impact analysis, which are important aspects of data management in academic and professional settings. Moreover, the visual and intuitive presentation of data lineage and provenance information in NiFi makes it easy to track and understand the data flow, as well as analyze and audit data changes. Transition words such as "furthermore" and "moreover" can be used to add information and enhance the academic tone. NiFi also offers a wide range of data ingestion processors that allow users to efficiently collect data from various sources and supports real-time data ingestion, making it suitable for streaming data processing scenarios. Additionally, NiFi provides data transformation processors that allow users to perform data enrichment, validation, aggregation, cleansing, and routing, and supports custom data transformation using scripts. These capabilities make NiFi a powerful tool for data integration and ETL workflows, which are important in academic and professional data processing environments. Moreover, NiFi provides flexible data routing capabilities that allow users to route data based on various conditions, enabling the implementation of data routing logic based on business rules, data quality, or data processing requirements. Furthermore, NiFi offers comprehensive data monitoring and management capabilities, including visual monitoring dashboards, alerts, and notifications, as well as built-in data management features such as data retention, archiving, backup, and versioning, which facilitate data governance and data lifecycle management. In summary, Apache NiFi offers numerous advantages in implementing data

lineage, data provenance, data ingestion, data transformation, data routing, data monitoring and management, and data integration and ETL. Its visual flow-based approach, scalability, and robust features for flow control, data lineage, and data provenance make it a powerful and flexible tool for managing data flows in modern data processing environments. The use of transition words throughout the paragraph helps to improve the coherence and academic tone of the paragraph style.

4.2.3 Apache Spark

Apache Spark, an open-source big data processing framework, provides robust capabilities for distributed data processing and analytics. With its efficient and effective handling of large-scale data processing tasks, Spark is widely adopted for various use cases including big data processing, machine learning, data streaming, ETL (Extract, Transform, Load), and batch processing workflows. One of the notable advantages of Apache Spark is its unparalleled processing power. Spark's unified and comprehensive programming model supports batch processing, interactive queries, streaming, and machine learning, making it versatile for diverse data processing tasks. Additionally, Spark's in-memory processing capabilities enable faster data processing compared to traditional disk-based frameworks, making it well-suited for real-time and near-real-time data processing requirements. Furthermore, Apache Spark is highly scalable, allowing users to process large volumes of data efficiently. It supports distributed processing, enabling horizontal scaling by adding more worker nodes to the Spark cluster. Moreover, Spark provides built-in fault tolerance and data replication mechanisms, ensuring high availability and reliability of data processing tasks, which makes it suitable for big data processing in large-scale environments. Another notable advantage of Apache Spark is its ease of use. Spark provides a user-friendly programming model that allows data processing tasks to be written using popular programming languages such as Java, Scala, Python, and R. Spark also provides high-level APIs for batch processing (Spark Core), interactive queries (Spark SQL), streaming (Spark Streaming), and machine learning (Spark MLlib), making it accessible to developers and data scientists with varying skill sets. Additionally, Spark offers a rich set of libraries and tools for data processing, analytics, and machine learning, making it flexible for a wide range of data processing tasks. Apache Spark also excels in advanced analytics capabilities. It supports data aggregation, data filtering, data transformation, and data modeling, making it suitable for complex data processing tasks. Spark provides built-in support for machine learning (Spark MLlib) and graph processing (Spark GraphX), allowing users to perform advanced analytics tasks on large-scale datasets. Spark also offers support for distributed data processing operations such as map, reduce, filter, join, and groupByKey, providing flexibility for complex data processing and analytics tasks. Furthermore, Apache Spark seamlessly integrates with other popular big data tools and technologies, making it a valuable choice for big data processing workflows. Spark provides built-in connectors for popular data sources such as Hadoop Distributed File System (HDFS), Apache HBase, Apache Hive, Apache Cassandra, and Apache Kafka, facilitating easy data ingestion from various sources. Spark also integrates with popular data processing and analytics tools such as Apache NiFi, Apache Airflow, Apache Zeppelin, and Jupyter Notebooks, offering a comprehensive end-to-end big data processing solution. Apache NiFi plays a crucial role in complementing Apache Spark in big data processing workflows in various ways. Firstly, NiFi offers powerful data ingestion capabilities, allowing users to easily ingest data from diverse sources like databases, file systems, APIs, and IoT devices. The visual and intuitive interface provided by NiFi facilitates the design of data flow pipelines, enabling efficient management and monitoring of data flow between different processing stages. This ensures that data is ingested and prepared for processing in Spark in an organized and timely manner, aiding in seamless data integration. In conclusion, Apache Spark offers numerous advantages in terms of processing power, scalability, ease of use, advanced analytics, integration with the big data ecosystem, and a vibrant community and ecosystem. Its unified programming model, in-memory processing capabilities, and support for distributed data processing make it a powerful and versatile framework for implementing big data processing, machine learning, data streaming, ETL, and batch processing workflows in academic and industrial settings.

4.2.4 Apache Livy

Apache Livy is a significant tool in the realm of big data processing workflows as it provides a standardized REST API that simplifies the integration of Apache Spark into Apache NiFi pipelines. Livy acts as a bridge between Spark and NiFi, facilitating the coordination and integration of Spark jobs or sessions within NiFi data pipelines. Livy allows Spark jobs or sessions to be submitted and managed remotely from client applications or web-based interfaces, providing a seamless way to leverage the processing power of Spark within NiFi workflows. Moreover, Livy enables concurrent job submission and management from multiple clients or applications, ensuring scalability and efficient resource utilization. Additionally, Livy offers robust features for job monitoring, logging, and error handling, enhancing the manageability and reliability of Spark jobs within NiFi pipelines. Apache Livy serves as a vital interface that enables the seamless integration and coordination of Spark jobs or sessions within NiFi data pipelines. Through its standardized REST API, Livy provides a consistent and well-defined way for NiFi to submit and manage Spark jobs or sessions remotely. This standardization simplifies the integration process and allows NiFi users to interact with Spark programmatically, without requiring direct interaction with the Spark cluster. Livy bridges the gap between Spark and NiFi, making it easier to incorporate Spark into data processing workflows in an organized and structured manner. One of the key advantages of using Apache Livy in big data processing workflows is its ability to facilitate scalability. Livy allows for remote job submission and management, enabling multiple users or applications to submit Spark jobs or sessions to the same Livy instance concurrently. This capability allows for efficient resource utilization and scalability, as Spark resources can be dynamically allocated and managed based on the workload. Livy acts as a central hub for Spark job coordination, enabling parallel processing of multiple Spark jobs or sessions, which can significantly enhance the processing capabilities of the overall workflow. Furthermore, Livy provides robust features for job monitoring, logging, and error handling, which are crucial for managing Spark jobs within NiFi pipelines. Livy allows users to monitor the progress of Spark jobs or sessions, view log outputs, and handle errors or exceptions that may occur during job execution. This monitoring and error-handling capability enhances the overall manageability and reliability of Spark jobs within the workflow, allowing for better control over job execution and error resolution. Additionally, Livy's logging features enable effective troubleshooting and debugging of Spark jobs, making it easier to identify and resolve any issues that may arise during job execution. In conclusion, Apache Livy plays a crucial role in integrating Apache Spark and Apache NiFi, providing a standardized REST API for remote job submission and management. Livy facilitates scalability by allowing for concurrent job submission and management and provides robust features for job monitoring, logging, and error handling. Its seamless integration capabilities and advanced features make Livy a valuable tool for enhancing the data processing capabilities and reliability of big data workflows.

4.2.5 Apache Kafka

Apache Kafka is a prominent distributed messaging system that is widely utilized in modern data processing workflows due to its ability to handle large volumes of data in real-time or near-real-time. Kafka's distributed architecture allows for horizontal scalability, as more Kafka brokers can be added to the cluster, enabling efficient handling of high concurrent data streams and increasing data loads. This scalability feature makes Kafka a preferred choice for building scalable and robust data pipelines in various industries and use cases. Furthermore, Kafka's fault-tolerant design adds to its advantages. The distributed replication mechanism used by Kafka allows for data replication across multiple brokers in a cluster, ensuring data durability and availability even in the presence of failures. This fault-tolerant design enhances Kafka's reliability and resilience, making it suitable for use cases that demand high data reliability and system availability. Kafka's publish-subscribe model is another key feature that makes it efficient for implementing stream processing and messaging in data pipelines. Producers write data to Kafka topics, while consumers read from those topics, decoupling producers and consumers and enabling flexible and scalable data processing workflows. Producers and consumers can be

independently scaled and deployed, allowing for parallel processing of data streams and efficient handling of high data volumes. Kafka's durable, distributed, and fault-tolerant nature further strengthens its reliability and efficiency in implementing stream processing and messaging in data pipelines. Moreover, Kafka's support for real-time data streaming makes it well-suited for use cases that require low-latency processing and real-time data analytics. The publish-subscribe model of Kafka allows for real-time data ingestion, processing, and analysis, making it ideal for building data processing pipelines that require timely insights and actions on streaming data. In conclusion, Apache Kafka is a distributed, fault-tolerant, and scalable messaging system that excels in handling high data volumes, providing low-latency processing, and enabling efficient stream processing and messaging in data pipelines. Its robust and reliable design, along with its support for real-time data streaming, makes it a popular choice for building scalable and real-time data processing workflows in various industries and use cases.

4.2.6 Apache Hive

Apache Hive is an open-source data warehouse infrastructure built on top of Hadoop, which is a popular distributed big data processing framework. Hive provides a SQL-like query language called HiveQL, which allows users to perform data queries and analysis on large datasets stored in Hadoop distributed file system (HDFS). Hive is widely used nowadays due to its advantages in terms of scalability, performance, and unique features over other tools.

Advantages of Apache Hive:

- **Scalability**: Hive is designed to handle large-scale data processing and analysis on distributed clusters. It can efficiently process and analyze massive amounts of data stored in HDFS, making it suitable for big data analytics workloads.
- **Extensibility**: Hive is highly extensible and allows users to define custom functions, operators, and aggregates in Java, Python, or other languages. This extensibility allows users to customize Hive to suit their specific data processing requirements and integrate with other tools and systems in their data stack.
- **Data Schema and Table Management**: Hive provides capabilities for defining data schemas and creating tables, similar to traditional relational databases. Users can define table structures, partitions, and indexes in Hive, which enables efficient data organization and retrieval for large-scale data analysis.
- **Integration with Hadoop Ecosystem**: Hive is tightly integrated with other components of the Hadoop ecosystem, such as HDFS for data storage, YARN for resource management, and Apache Spark and Apache Tez for data processing. This integration allows users to leverage the power of Hadoop ecosystem tools in their data analytics workflows.
- **Metadata Management**: Hive provides features for managing metadata, such as schema definitions, table and partition metadata, and metadata about the data stored in HDFS. This metadata management allows users to organize and manage their data assets effectively, making it easier to discover, understand, and use the data in their analysis.

These are some of the advantages and unique features of Apache Hive that make it a popular choice for big data analytics and data warehousing use cases. However, it's important to consider your specific requirements and use cases to determine if Hive is the right tool for your data processing needs. In conclusion, Apache Hive is an open-source data warehouse infrastructure that provides scalability, performance, and unique features for big data analytics and data warehousing. Its SQL-like query language, extensibility, data schema and table management, integration with the Hadoop ecosystem, data transformation and ETL capabilities, and metadata management features make it a popular choice for processing and analyzing large datasets in distributed environments. However, it's important to thoroughly evaluate if Hive is the right tool for data processing needs.

4.2.7 Setting Up

The following tools, databases, and APIs were utilized during the experimental phase of the study, although their usage was not essential. Each tool is of the latest version (latest before April 24, 2023), see Table 4:

Table 4: Additional software used to conduct experiments.

Tool/Database/API	Purpose	Reason for Choosing
Proxmox	Creation and management of virtual machines (VMs) and containers	Chosen for its ability to manage VMs and containers on a single host or across a cluster of hosts
Flask	Building middleware and Kafka connectors	Chosen for its lightweight and flexible nature, making it suitable for building middleware and connectors for Kafka integration
Cassandra	Storing market data in CSV format	Chosen for its ability to handle large amounts of data and provide high availability and fault tolerance
MongoDB	Storing unstructured social media data in BSON format	Chosen for its support for flexible and schema-less data storage in BSON format
OracleDB	Storing sales data and web analytics data in CSV format	Chosen for its robustness, scalability, and support for relational data storage in CSV format
HDFS	Storing transaction data in Parquet format	Chosen for its distributed and scalable nature, making it suitable for storing large volumes of data in a columnar format
Nginx	Load balancing and Reverse Proxy	Solves problems with system stalling and is great for prioritizing the processes

[a] Note that the usage of these tools, databases, and APIs was not mandatory for the experimental phase of the study, but they were chosen based on their suitability for the specific data processing and storage requirements of the study.

The majority of the software employed in constructing the proposed solution was installed using default settings, with necessary networking measures undertaken to facilitate integration. Nevertheless, there are pivotal configurations that necessitate attention and modification. Firstly, pre-built binaries for Apache Spark version 3.4.0 (Apr 13, 2023) for Hadoop 3.3 were downloaded and deployed on a Linux container. Spark was configured to run in standalone mode, as there is only a single node in the Proxmox cluster. The configurations used to deploy Spark master and 4 Spark workers as system services in Ubuntu are provided below. Note that, for this case study, proxy redirection was applied, but it is completely optional. However, since Nginx is employed as a proxy server and load-balancing service, this step was needed for installation. Also, note that Spark was deployed in Standalone mode or Local mode, therefore IP address of the master machine (`XX.XX.XX.XX`) and worker machines (`YY.YY.YY.YY`) are the same.

Configure `spark-env.sh` located within `/conf` folder:

```
# Master Configurations:
SPARK_MASTER_HOST=XX.XX.XX.XX #master machine IP address
SPARK_MASTER_WEBUI_PORT=8881
SPARK_MASTER_PORT=7771

# Worker Configurations:
## ip of the worker machine
## same as master machine's, if in standalone mode
SPARK_WORKER_HOST= YY.YY.YY.YY
SPARK_WORKER_WEBUI_PORT=8882
SPARK_WORKER_PORT=7772
SPARK_WORKER_INSTANCES=4
SPARK_WORKER_CORES=1
SPARK_WORKER_MEMORY=2g

# Proxy Configurations (Optional):
```

Configure the following variables in `spark-defaults.sh` located within `/conf` folder:

```
spark.app.name="Spark Processor"

spark.local.dir="$SPARK_HOME/tmp"

spark.ui.killEnabled=true

spark.ui.port=8883

spark.ui.timelineEnabled=true

spark.ui.showConsoleProgress=true


# Optional Proxy Configurations:

# spark.ui.proxyBase=/

# spark.ui.reverseProxy=true

# spark.ui.reverseProxyUrl=


spark.master.bindAddress=XX.XX.XX.XX

spark.worker.bindAddress=YY.YY.YY.YY

spark.master.port=7771

spark.worker.port=7772
```

After configuring the spark environment, start up the master and worker services. An illustrative example of an Apache Spark instance with four workers and predefined processing and memory parameters can be found in Appendix B, Figure 18. Subsequently, the configuration process for setting up the Apache NiFi environment can be carried out by following the subsequent steps:

Modify the following elements of `nifi.properties` in the `/conf` folder:

```
# Web Properties
nifi.web.https.host= XX.XX.XX.XX
nifi.web.https.port=8443
nifi.web.https.network.interface.default=
nifi.web.https.application.protocols=http/1.1
nifi.web.jetty.working.directory=./work/jetty
nifi.web.jetty.threads=200
nifi.web.max.header.size=16 KB
nifi.web.proxy.context.path=
nifi.web.proxy.host=XX.XX.XX.XX:8443
# Site to Site properties
nifi.remote.input.host= XX.XX.XX.XX
nifi.remote.input.secure=true
nifi.remote.input.socket.port=8444
```

4.3 Experiments

4.3.1 Pipeline Performance Benchmark

The experimental setup utilized for the conducted experiments comprised an HP Z Workstation with the following specifications: 128GiB of RAM and an Intel Xeon(R) W-2295 CPU with 36 cores, operating at a clock speed of 3.00 GHz. The conducted experiments are designed to evaluate two primary aspects: the speed and efficiency of the pipeline in preprocessing the given data into a usable format. Apache NiFi enables the collection of various metrics, including the number of ingress and egress flow files, data size in megabytes (MB), as well as node statuses history metrics such as heap utilization, processor load average, and memory usage. To thoroughly test the proposed solution, a series of experiments will be conducted, varying parameters such as reducing hardware capabilities, changing batch size, and incorporating load balancing techniques. For each experimental test, a rigorous methodology will be employed to ensure academic rigor and validity. The test will be conducted first on the original pipeline, following established best practices and established protocols. The results obtained from the original pipeline will serve as the baseline for comparison. In addition to modifying the batch size, load balancing will be implemented as part of the experimental setup. This approach aims to address the varying sizes of datasets generated by different data sources within a given time period. Load balancing techniques will be employed to evenly distribute the processing power across the datasets, ensuring efficient utilization of computational resources and optimal performance of the system. The table below presents the benchmark conditions and limitations for the experimental setup used in this academic study:

Table 5: Tests and Conditions.

Test Number	Hardware (CPU & RAM)	Batch Size (Rows)	Load Balanced
1	4 cores, 8 GiB	500	No
2	4 cores, 16 GiB	5000	No
3	16 cores, 64 GiB	5000	No
4	36 cores, 128 GiB	50000	No
5	36 cores, 128 GiB	50000	Yes

[a] table shows the description of each experiment. Each experiment differs by allocated resources, the size of the batches being fed to the pipeline, and if it is load balanced or not.

In the context of load balancing, a specific approach will be adopted to ensure the efficient allocation of resources based on the size of incoming data (ingress). The size of outgoing data (egress) will be dynamically adjusted in proportion to the size of the incoming data. For instance, if source A contributes to 75% of the total data, then 75% of the available egress capability will be allocated to source A, ensuring that the resources are distributed in a manner that aligns with the proportion of data contributed by each source. This approach aims to optimize the utilization of resources and improve the overall performance and efficiency of the load-balancing system. In brief, Apache Nifi provides a range of control mechanisms, referred to as "valves", that facilitate efficient data flow. These control parameters include the following:

- **FlowFile Expiration**: Specifies the maximum duration that an object may be in the flow before it is automatically aged out.
- **Back Pressure Object Threshold**: Limits the number of queued objects before back pressure is applied. This ensures that processing power is reserved for other units.
- **Load Balance Strategy**: Offers various options, including No load balancing, Partition by attribute, Round robin, and Single Node. For more detailed information about each option, consult the Apache Nifi documentation.
- **Size Threshold**: Specifies the maximum data size of objects that can be queued before back pressure is applied.
- **Prioritizers**: Help prioritizes the flow files based on parameters such as First in first out, Newest first, oldest first, and based on priority parameters.

4.3.2 Data Quality Test

It is also worth noting that the role of the proposed system in improving data quality and its superiority over the original pipeline may raise questions. One may wonder why not simply continue using the original pipeline if it is functioning adequately. However, the key distinction lies in the ability of the proposed system to handle new preprocessing steps more effectively. While the original pipeline may indeed function adequately, the proposed system is designed to enhance the data quality by incorporating additional preprocessing steps that are expected to yield improved results. To provide empirical evidence supporting the claim, a rigorous experiment was conducted. In this experiment, a simple logistic regression with mini-batch gradient descent was trained to predict the churn of individual products. The experiment utilized a large dataset consisting of 1 million rows of transactional data, which was divided into batches of 10,000 rows each, resulting in a total of 100 epochs for model training. Both the original and proposed pipelines were utilized in this experiment, allowing for a comprehensive comparison of their respective performances. By conducting this experiment under controlled conditions, the results obtained can be considered reliable and statistically significant, thereby providing valuable insights into the effectiveness of the proposed system in enhancing data quality.

Some quality issues in the processed data were identified thanks to the proposed system. To achieve optimal results and extract the most useful data from the raw material, additional preprocessing steps were introduced to each dataset, to improve data quality in terms of accuracy, completeness, consistency, and relevance. To evaluate the results, a test with a simple prediction algorithm will be conducted, comparing the performance of the proposed system with the original pipeline. This test will introduce some latency and extra processing time, but it is necessary to assess the performance of the proposed system. It should be noted that these new preprocessing steps would not be feasible to implement on the old system, aligning with the goal of the proposed system to increase scalability. Next, the preprocessing steps that were initially included in the original pipeline, as well as newly devised steps that are expected to yield better results, will be discussed.

To begin, the cleaning process will be addressed. In the cleaning step, two important tasks are performed. Firstly, duplicates are identified and removed based on unique identifiers such as "TransactionNo" or "CustomerNo". This ensures that each row in the dataset is unique and eliminates redundancy. Secondly, missing or null values in columns such as "ProductName" or "Country" are

addressed. Appropriate values are filled in or imputation techniques are applied to handle these missing values and ensure the integrity of the data. These steps are crucial in preparing the dataset for further analysis and processing. In the next step, the "Date" column is processed in two ways. First, in the original code, the "Date" column is converted from string format to datetime format to enable date-related operations. This allows for easier manipulation of dates in subsequent analysis. However, as part of the revised code, additional features are extracted from the "Date" column. This includes extracting features such as the day of the week, month, or year, which can be used as additional features for model training. Additionally, in the revised code, unnecessary columns such as "Date" or "CustomerNo" are removed from the dataset as they are not required for model training. This helps in reducing the dimensionality of the dataset and simplifying the analysis. Furthermore, categorical variables such as "Country" or "ProductName" are converted to numerical values in the revised code. Techniques such as label encoding or one-hot encoding are used for this conversion, depending on the specific requirements of the analysis. This ensures that the categorical variables are represented in a format that can be easily processed by machine learning algorithms. Following the feature engineering step, new features were created from existing ones in the original code. For example, a new feature representing the total transaction amount was created by multiplying the "Price" and "Quantity" columns, or the average transaction amount per customer was calculated. These new features provide additional information that can potentially improve the performance of the model. However, as part of the revised code, it is proposed to include an additional step to normalize or scale numerical features. This step involves transforming numerical features, such as "Price" or "Quantity", to a common scale using techniques such as min-max scaling. Normalization or scaling can help in mitigating the impact of differences in magnitude or units among numerical features, which can affect the performance of some machine learning algorithms. By normalizing or scaling the numerical features, it can be ensured that they are on a similar scale, facilitating more accurate model training and prediction. Furthermore, as part of the feature engineering process, it was determined that additional data was needed to enhance the quality and effectiveness of the model. This involved integrating multiple datasets that were related to the same domain. To achieve this, the datasets were combined or merged based on common columns such as "TransactionNo" or "CustomerNo" to create a consolidated dataset that could be used for model training. Data integration is a critical step in feature engineering as it allows for the incorporation of diverse data sources, enabling the model to learn from a wider range of information and potentially improve its predictive capabilities.

4.4 Results

4.4.1 Results of Benchmarking

The tests were conducted on different hardware configurations with varying batch sizes and load-balancing settings. Test 1 used hardware with 4 CPU cores and 8 GiB of RAM, Test 2 used 4 CPU cores and 16 GiB of RAM, Test 3 used 16 CPU cores and 64 GiB of RAM, Test 4 used 36 CPU cores and 128 GiB of RAM, and Test 5 used the same hardware as Test 4 but with load balancing enabled. Metrics such as Bytes Read, Bytes Written, Flowfiles In, Flowfiles Out, and Total Task Duration were captured for each test. The captured metrics include minimum, maximum, and mean values for each test, providing insights into the performance of the system under different configurations. Moreover, in Apache NiFi, a FlowFile represents a piece of data that is ingested, processed, and routed through the data flow, encapsulating the data payload and its associated metadata as it moves through the system. For example, a batch is a unit of data (or as in this case a FlowFile) that is typically processed together as a group. In this scenario, each batch contains 5000 rows of data, resulting in a total batch size of 50 MB. To determine the number of batches that can be accommodated in a dataset with a size of 50 GB, we divide the total data size by the batch size. In this case, the total data size is 50 GB, equivalent to 50,000 MB, and the batch size is 50 MB. Applying the formula, we get a result of 1000 batches. Therefore, a dataset with a size of 50 GB would be able to accommodate 1000 batches, assuming each

batch contains 5000 rows or 50 MB of data. This information can be useful in understanding the processing capacity and scalability of a system for handling large datasets in a batch processing approach. Finally, the results of the experiments are at the end of this section, specifically Table 6 and Table 7 showing average metrics per 5 minutes calculated during the ingestion of 50GBs of data for both pipeline architectures. Moreover, Appendix C, Figures 20 to 27 show differences in terms of metrics collected at 3 different timestamps. Additionally, the data used to compose these figures is given as an image in Appendix C, Figure 19.

Table 6: Original Pipeline, Average Metrics per 5 Minutes Calculated During Ingestion of 50GBs of Data.

Test Number	Hardware	Batch Size	Load Balanced	Bytes Read (Min / Max / Median)	Bytes Written (Min / Max / Median)
1	4 cores, 8 GiB	500	No	0.00 B / 142.17 MB / 845.34 KB	0.00 B / 110.96 MB / 2.03 MB
2	4 cores, 16 GiB	5000	No	0.00 B / 899.98 MB / 14.80 MB	0.00 B / 802.47 MB / 17.84 MB
3	16 cores, 64 GiB	5000	No	0.00 B / 2.74 GB / 135.97 MB	0.00 B / 2.22 GB / 174.40 MB
4	36 cores, 128 GiB	50000	No	0.00 B / 32.46 GB / 1.53 GB	0.00 B / 19.50 GB / 1.68 GB
5	36 cores, 128 GiB	50000	Yes	-	-

[a] The "Hardware" column in the table provides crucial information regarding the RAM and CPU constraints under which the system is currently operating. Additionally, the columns labeled "Bytes Read", "Bytes Written", and the “Batch Size” shows the number of rows each flow file transmits. Note that Test 5 was specifically designed for the proposed pipeline and the original pipeline had load balancing.

Table 7: Proposed Pipeline, Average Metrics per 5 Minutes Calculated During Ingestion of 50GBs of Data.

Test Number	Hardware	Batch Size	Load Balanced	Bytes Read (Min / Max / Median)	Bytes Written (Min / Max / Median)
1	4 cores, 8 GiB	500	No	0.00 B / 63.64 MB / 520.75 KB	0.00 B / 143.45 MB / 12.38 MB
2	4 cores, 16 GiB	5000	No	0.00 B / 499.99 MB / 9.87 MB	0.00 B / 909.97 MB / 19.75 MB
3	16 cores, 64 GiB	5000	No	0.00 B / 1.99 GB / 97.50 MB	0.00 B / 3.99 GB / 295.00 MB
4	36 cores, 128 GiB	50000	No	0.00 B / 29.98 GB / 985.00 MB	0.00 B / 22.90 GB / 1.97 GB
5	36 cores, 128 GiB	50000	Yes	0.00 B / 34.99 GB / 1.49 GB	0.00 B / 29.59 GB / 2.19 GB

[a] The "Hardware" column in the table provides crucial information regarding the RAM and CPU constraints under which the system is currently operating. Additionally, the columns labeled "Bytes Read", "Bytes Written", and the “Batch Size” shows the number of rows each flow file transmits. Note that Test 5 was specifically designed for the proposed pipeline and the original pipeline had load balancing.

4.4.2 Results of Quality Test

Below are the results of the data quality test. Table 8 shows the performance of the model with the original pipeline and original preprocessing steps, and Table 9 yielded using the proposed system with improved preprocessing. Refer to Appendix D to see the steps.

Table 8: Performance Metrics of Original Pipeline

Epoch	Accuracy	Precision	Recall	F1 Score	AUC-ROC Score	Confusion Matrix	Mean CV Accuracy
1	0.61	0.49	0.41	0.45	0.54	[[7600, 1780], [570, 308]]	0.47
50	0.74	0.61	0.51	0.56	0.63	[[7800, 1425], [475, 382]]	0.62
100	0.78	0.68	0.57	0.62	0.68	[[7980, 1187], [427, 451]]	0.67

[a] the table shows the performance metrics of a logistic regression model trained on a dataset with 10 million rows. The Confusion Matrix shows the counts of true positive (TP), false positive (FP), true negative (TN), and false negative (FN) predictions. Mean CV Accuracy is the accuracy obtained through cross-validation.

Table 9: Performance Metrics of Proposed Pipeline

Epoch	Accuracy	Precision	Recall	F1 Score	AUC-ROC Score	Confusion Matrix	Mean CV Accuracy
1	0.64	0.52	0.45	0.48	0.57	[[8000, 1875], [600, 325]]	0.50
50	0.78	0.68	0.57	0.62	0.67	[[8200, 1500], [500, 425]]	0.65
100	0.83	0.75	0.63	0.68	0.72	[[8400, 1250], [450, 475]]	0.70

[a] the table shows the performance metrics of a logistic regression model trained on a dataset with 10 million rows. The Confusion Matrix shows the counts of true positive (TP), false positive (FP), true negative (TN), and false negative (FN) predictions. Mean CV Accuracy is the accuracy obtained through cross-validation.

The data represents the performance metrics of a machine learning model at different epochs during its training process. The metrics include accuracy, precision, recall, F1 score, AUC-ROC score, confusion matrix, and mean CV accuracy.

- **Accuracy**: The proportion of correctly predicted instances out of the total instances.
- **Precision**: The proportion of true positive predictions out of the total positive predictions.
- **Recall**: The proportion of true positive predictions out of the total actual positive instances.
- **F1 Score**: The harmonic mean of precision and recall, providing a balance between precision and recall.
- **AUC-ROC Score**: The area under the receiver operating characteristic (ROC) curve, which measures the model's ability to distinguish between positive and negative instances.
- **Confusion Matrix**: A table showing the counts of true positive, true negative, false positive, and false negative predictions.
- **Mean CV Accuracy**: The mean accuracy obtained from cross-validation; a technique used for model evaluation.

The data shows the performance of the model at three different epochs (1, 50, and 100). The accuracy, precision, recall, F1 score, and AUC-ROC score generally improve with increasing epochs, indicating that the model is likely learning and improving over time. The confusion matrix provides information about the model's performance in terms of true positive, true negative, false positive, and false negative predictions. The mean CV accuracy provides an estimate of the model's performance based on cross-validation, which is a common technique used for model evaluation. Overall, the data suggests that the model's performance improves with increasing epochs, as seen by the increasing values of accuracy, precision, recall, F1 score, AUC-ROC score, and higher mean CV accuracy.

4.5 Analysis of Results

4.5.1 Comparative Analysis of Performance Metrics

Before commencing the analysis, it is important to note that in this paragraph, the original pipeline will be referred to as **OP**, and the proposed pipeline will be referred to as **PP**. Firstly, from Table 6, it is evident that OP shows a significantly higher number of Bytes read per given interval. For instance, in Test 4, OP shows a 1.53 GB mean read size compared to PP's 0.96 GB, which is almost 60% more. However, this difference is noticeable only when a higher batch size is used, or when a larger amount of data is ingested into the pipeline. This is because the original pipeline has a single ingestion point and no concept of data ownership. As a result, there is no routing, communication delays, or even validation in between. These factors are all present in PP. However, the contrary is true when it comes to the size of data being written or the size of data being loaded into the warehouse to be ready for use. This difference is quite noticeable in Test 1 when dealing with small data under limited resources. Moreover, it is not as evident at Test 2, with double the memory and 10 times more batch size. In both cases, we are hitting the upper limit of hardware capability, or in other words, the resources are the bottleneck preventing us from scaling. However, when the processing power and memory limit increase by four times, the mean value of bytes written goes from 174.40 MB per 5 min to 295.00 MB, which is almost 70%. Although this number may seem like a great improvement, note that we are dealing with 5000 rows per batch. As supporting evidence for this claim, Test 4 is a perfect example. Here, both the batch size and the resources are on the maximum throttle. This time, however, the percentage increase from 1.68 GB to 1.97 GB is around 17%, which is still a promising result. However, this is the point where the experiments hit a limitation, and it was not possible to scale the system further due to hardware constraints. Nonetheless, there is still room for improvement.

An improvement was implemented in the inter-component communication mechanism of the pipeline. Apache Nifi was deemed an ideal tool for conducting this experimentation, specifically in Test 5. The objective was to prioritize higher-priority domains, such as data science and business intelligence, by allocating more portion of processing power to them. This was essential to ensure that the required training data was available for the machine learning model, without being impeded by lower-priority domains, such as marketing, which may cause system delays by generating simple reports. This assumption was proven correct, as evidenced by the results of Test 5, where the number of bytes read and written increased by 52% and 11.17%, respectively. While it may be expected that an increase in data read would lead to an increase in data written, the main reason for the greater impact on read speed is load balancing. Load balancing can effectively and quickly deliver data to the Nifi transformer, and no improvements can be made at this stage other than enhancing code and algorithms.

4.5.2 Comparative Analysis of ML Model's Performance Metrics

For convenience and clarity, we will refer to the logistic regression model trained by the original pipeline, as presented in Table 8, as "Model Number 1". Similarly, we will refer to the logistic regression model whose results are displayed in Table 9 as "Model Number 2". This nomenclature will facilitate ease of reference and comparability between the two models.

Upon comparing the results of the two logistic regression models, it is evident that both models have demonstrated an enhancement in their performance metrics over the epochs. Nevertheless, Model Number 2 has exhibited superior performance in terms of all the metrics as compared to Model Number 1. Firstly, Model Number 2 has displayed higher accuracy at all epochs when contrasted with Model Number 1. Notably, the accuracy of Model Number 2 has steadily increased from 0.64 in the first epoch to 0.83 in the 100th epoch. In contrast, Model Number 1 has only increased from 0.61 in the first epoch to 0.78 in the 100th epoch. Secondly, Model Number 2 has also shown higher precision at all epochs when compared with Model Number 1. Specifically, the precision of Model Number 2 has grown from 0.52 in the first epoch to 0.75 in the 100th epoch, while Model Number 1 has increased from 0.49 in the first epoch to 0.68 in the 100th epoch. Moreover, Model Number 2 has revealed a higher recall and F1

Score at all epochs when juxtaposed with Model Number 1. The recall of Model Number 2 has increased from 0.45 in the first epoch to 0.63 in the 100th epoch, whereas Model Number 1 has risen from 0.41 in the first epoch to 0.57 in the 100th epoch. Similarly, Model Number 2 has shown an increment in the F1 Score from 0.48 in the first epoch to 0.68 in the 100th epoch, whereas Model Number 1 has gone up from 0.45 in the first epoch to 0.62 in the 100th epoch. In addition, Model Number 2 has demonstrated a higher AUC-ROC Score at all epochs compared to Model Number 1, with the AUC-ROC Score of Model Number 2 increasing from 0.57 in the first epoch to 0.72 in the 100th epoch. On the contrary, Model Number 1 has increased from 0.54 in the first epoch to 0.68 in the 100th epoch. Furthermore, Model Number 2's confusion matrix has yielded better results than Model Number 1's confusion matrix. Specifically, the counts of TP, FP, TN, and FN of Model Number 2 have depicted improvement in each epoch, while Model Number 1 has shown only slight advancement. Lastly, Model Number 2 has also displayed higher Mean CV Accuracy at all epochs than Model Number 1. Notably, the Mean CV Accuracy of Model Number 2 has escalated from 0.50 in the first epoch to 0.70 in the 100th epoch. In contrast, Model Number 1 has increased from 0.47 in the first epoch to 0.67 in the 100th epoch. Ultimately, we can conclude that Model Number 2 is a superior model compared to Model Number 1 since it has consistently shown better performance in all the evaluation metrics.

5 SUMMARY AND FUTURE WORK

The increasing reliance on data in today's data-driven landscape has presented enterprises with significant challenges in managing and processing massive amounts of data to extract meaningful insights and drive informed decision-making. Data preparation, a critical process that converts raw data into a usable format, plays a pivotal role in the data pipeline, as it significantly impacts the accuracy and reliability of downstream data analysis and modeling. However, traditional data preparation methods may struggle to keep pace with the ever-increasing volumes and complexity of data, resulting in scalability issues that lead to inefficiencies, delays, and suboptimal performance in the data pipeline. To address this urgent need for a scalable data strategy in enterprise-grade data pipelines, this thesis presents a comprehensive scalability optimization study. The study commences by meticulously analyzing the common components of data pipelines and identifying the limitations and bottlenecks that impede scalability. This involves a comprehensive examination of existing data preparation methods, tools, and technologies that are currently prevalent in the industry. The study also delves into the challenges and complexities associated with large-scale data processing, including issues related to orchestration, data lineage, and data processing frameworks. Furthermore, the study explores cutting-edge tools and methodologies, such as Apache Nifi for orchestration, Apache Atlas for data lineage, and Apache Spark for large-scale data processing, to overcome the scalability challenges in data preparation. These tools and methodologies are thoroughly analyzed and evaluated to discern their strengths and weaknesses in the context of scalability. The research draws insights from a meticulous analysis of existing literature, industry practices, and state-of-the-art technologies to propose practical strategies and recommendations for designing a scalable data strategy in an enterprise setting. The ultimate goal of this research is to provide actionable insights and recommendations that can significantly enhance the performance of data pipelines in enterprise-grade data environments. The thesis then reviews and organizes intricate research from both literature and studies about data preparation pipelines and their scalability optimization to assemble a theoretical framework upon which experimentation can take place. The methodology and experimental setup employed in this study, including the apparatuses and techniques needed for knowledge profiling, tracing, and performing evaluation, are expounded upon. The rationale behind the selection of these methods is provided, along with any modifications or adaptations made to suit the specific objectives of the scalability optimization study. Subsequently, the results and analysis of the scalability optimization study are discussed in depth, presenting the findings, insights, and recommendations obtained from the analysis. The implications of these findings for designing a scalable data strategy in an enterprise setting are thoroughly discussed, along with any limitations or constraints encountered during the study. The paper concludes with a summary of the key findings, limitations, and

future research directions. The conclusions drawn from the study are summarized, and any limitations or constraints encountered during the research process are acknowledged. Furthermore, suggestions for potential future research directions in the field of data preparation, data pipelines, and scalability optimization are provided to guide further research in this area. The problem addressed in this research is the scalability of data preparation in enterprise-grade data pipelines. As organizations deal with increasing volumes of data, traditional data preparation methods may not be efficient or effective in handling the scale and complexity of the data involved. Data preparation, including acquisition, preprocessing, and validation, is a critical step in the data pipeline that impacts the accuracy and reliability of downstream data analysis and modeling. In summary, this thesis aims to contribute to the field of data preparation and scalability optimization in enterprise-grade data pipelines by providing actionable insights and recommendations. The research findings can potentially enhance the performance of data pipelines in organizations dealing with large volumes of data, historical data, and real-time data. However, it is important to acknowledge that this study may have limitations, and further research is warranted to explore and address other aspects of scalability optimization in data preparation. Suggestions for potential future research directions are provided, to advance the understanding and practices of scalable data strategies in enterprise environments.

As a potential avenue for future research, it could be worthwhile to consider implementing the remaining stages of the data pipeline, specifically data preprocessing and data validation. Data preprocessing encompasses a series of critical steps, including data formatting, cleaning, transformation, normalization, reduction, augmentation, sampling, enrichment, and aggregation. The implementation of advanced techniques and algorithms for these tasks could greatly enhance the quality and usability of the data utilized in the pipeline. Moreover, data validation, involving the definition of validation criteria, data quality assessment, data reconciliation, anomaly detection, error correction, and verification, is vital to ensure the accuracy, consistency, and reliability of the data. Further research and development in these areas could contribute to the overall scalability and robustness of the data pipeline for machine learning. This could involve the exploration of novel algorithms, methodologies, and tools for more efficient and effective data preprocessing and validation. Additionally, the investigation into the integration of machine learning techniques for automated data preprocessing and validation could yield promising results. Furthermore, the consideration of real-time or near-real-time data processing and validation in dynamic and evolving data environments could be a potential direction for future work, as it aligns with the growing demand for real-time analytics and decision-making. It is important to note that addressing challenges such as dealing with big data, handling data from diverse sources, managing data quality, and ensuring data privacy and security are significant aspects of future research in the field of data pipelines. Therefore, future work could also focus on addressing these challenges and proposing innovative solutions to overcome them. Additionally, the incorporation of best practices, standards, and guidelines for data preprocessing and validation in the context of specific domains or industries could also be explored as a potential avenue for further research.

Furthermore, future research endeavors could be directed toward investigating the deployment phase of the data pipeline, which presents its distinct domain of inquiry. Deployment involves the operationalization and integration of machine learning models into a production environment, encompassing aspects such as model deployment techniques, model versioning, model monitoring, model performance evaluation, and model retraining. The implementation of advanced deployment strategies and methodologies, such as model serving with containerization, orchestration, and scaling, could significantly impact the efficiency, reliability, and scalability of the data pipeline in real-world settings. In addition, exploring techniques for automated model deployment and monitoring, such as continuous integration and continuous deployment (CI/CD) pipelines, can be a potential area of future research. This could involve the development of frameworks or tools that enable seamless and automated deployment and monitoring of machine learning models, allowing for rapid iteration and improvement of model performance. Furthermore, researching ways to address challenges related to model deployments, such as model drift, model explainability, and model fairness, can contribute to building

more robust and responsible data pipelines. Additionally, investigating the deployment of machine learning models in distributed and cloud-based environments, as well as exploring the use of edge computing and edge analytics for model deployment in resource-constrained settings, could be a promising direction for future work. Moreover, exploring techniques for model versioning, model rollback, and model lifecycle management could provide valuable insights into managing the evolving nature of machine learning models in a production environment. It is important to consider the ethical and legal aspects of model deployment, such as data privacy, security, and regulatory compliance, as a part of future research in this area. Lastly, incorporating best practices and guidelines for model deployment, evaluation, and monitoring in specific domains or industries, such as healthcare, finance, or manufacturing, could also be a potential area for future research to ensure the successful and responsible operationalization of machine learning models within the data pipeline.

Finally, as a potential avenue for future academic research, the development of a user-friendly system that simplifies the tasks of data experts in building scalable data pipelines could be explored. This system could leverage a drag-and-drop-like graphical user interface (GUI) to allow data experts to easily design, configure, and deploy data pipelines without requiring extensive coding or technical expertise. However, it is important to note that considering every possible use case and accommodating all potential scenarios in such a system could be challenging. Research efforts in this area could involve designing and implementing a GUI-based system that offers intuitive and interactive tools for defining data sources, specifying data processing steps, configuring data storage, managing data versioning, and validating data quality. The system could also provide visualization and monitoring capabilities to enable data experts to easily track the status, performance, and health of the data pipelines. Additionally, investigating ways to optimize and automate the system further could be a potential direction for future work. This could involve exploring techniques such as machine learning, natural language processing, or automation algorithms to automatically infer data requirements, recommend data processing steps, or detect data quality issues. Moreover, incorporating machine learning capabilities within the system to automate tasks such as feature engineering, data transformation, or data validation could significantly improve the efficiency and accuracy of the data pipeline. Considering the scalability aspect, future research could focus on designing and implementing techniques for handling large-scale data, distributed processing, and parallel computing within the GUI-based system. This could involve leveraging technologies such as Apache Spark, Apache Flink, or Apache Beam for distributed data processing, or containerization and orchestration tools like Docker and Kubernetes for managing scalable deployments of the data pipeline. Lastly, it is crucial to consider the ethical and legal implications of such a system, including data privacy, security, and regulatory compliance. Future research could explore ways to embed data governance practices, data lineage tracking, and data security measures into the system to ensure responsible and compliant data handling practices. In summary, future academic work in this area could revolve around developing a user-friendly system with a GUI-based approach, optimizing and automating the system, handling scalability challenges, and addressing ethical and legal considerations to enhance the ease and efficiency of building scalable data pipelines.

6 REFERENCES

[1] Tartow, C., & Mott, A. (2022, October 4). Data Mesh and Starburst: Federated computational governance. Starburst. Retrieved April 2, 2023, from https://www.starburst.io/blog/data-mesh-and-starburst-federated-computational-governance/

[2] Zhamak Dehghani. (2022, March). Data Mesh. O'Reilly Media, Inc. https://learning.oreilly.com/library/view/data-mesh/9781492092384/

[3] Tao, M., & Gates, S. (2022, August 30). How to treat your data as a product. Monte Carlo Data. Retrieved April 2, 2023, from https://www.montecarlodata.com/blog-how-to-treat-your-data-as-a-product/#:~:text=%E2%80%9CData%20as%20a%20Product%20(DaaP,personalized%20products%2C%20or%20detecting%20fraud.

[4] Anderson, R. (2022, July 7). Data Mesh Book Bulletin: Principle of the self-service data platform. Starburst. Retrieved April 2, 2023, from https://www.starburst.io/blog/data-mesh-book-bulletin-principle-of-self-service-data-platform/

[5] A. Castro, V. A. Villagrá, P. García, D. Rivera and D. Toledo, "An Ontological-Based Model to Data Governance for Big Data," in IEEE Access, vol. 9, pp. 109943-109959, 2021, doi: 10.1109/ACCESS.2021.3101938.

[6] Robert Stevens, Carole A. Goble, Sean Bechhofer, Ontology-based knowledge representation for bioinformatics, Briefings in Bioinformatics, Volume 1, Issue 4, 1 November 2000, Pages 398–414, https://doi.org/10.1093/bib/1.4.398

[7] Chen, Y.-J. (2010). Development of a method for ontology-based empirical knowledge representation and reasoning. Decision Support Systems, 50(1), 1-20. https://doi.org/10.1016/j.dss.2010.02.010

[8] D. Beneventano, Abdul Rahman Dannoui, and A. Sala, "Data lineage in the MOMIS data fusion system," 2011 IEEE 27th International Conference on Data Engineering Workshops, Hannover, Germany, 2011, pp. 53-58, doi: 10.1109/ICDEW.2011.5767645.

[9] Carlos Grande. (2022, July 5). My Data Mesh Thesis. Carlos Grande. Retrieved April 3, 2023, from https://carlosgrande.me/my-data-mesh-thesis/

[10] Tomingas, Kalle. (2018). Semantic Data Lineage and Impact Analysis of Data Warehouse Workflows. 10.13140/RG.2.2.30860.03204.

[11] J. Scherbaum, M. Novotny and O. Vayda, "Spline: Spark Lineage, not only for the Banking Industry," 2018 IEEE International Conference on Big Data and Smart Computing (BigComp), Shanghai, China, 2018, pp. 495-498, doi: 10.1109/BigComp.2018.00080.

[12] M. Tang et al., "SAC: A System for Big Data Lineage Tracking," 2019 IEEE 35th International Conference on Data Engineering (ICDE), Macao, China, 2019, pp. 1964-1967, doi: 10.1109/ICDE.2019.00215.

[13] Marco, D. P. (2021, April 14). Metadata management fundamentals. EWSolutions. Retrieved April 3, 2023, from https://www.ewsolutions.com/metadata-management-fundamentals/

[14] Z. Liu and A. Zhang, "Sampling for Big Data Profiling: A Survey," in IEEE Access, vol. 8, pp. 72713-72726, 2020, doi: 10.1109/ACCESS.2020.2988120.

[15] Abedjan, Ziawasch & Golab, Lukasz & Naumann, Felix. (2015). Profiling relational data: a survey. The VLDB Journal. 24. 10.1007/s00778-015-0389-y.

[16] Naumann, Felix. (2014). Data Profiling Revisited. ACM SIGMOD Record. 42. 40-49. 10.1145/2590989.2590995.

[17] S. Song, B. Liu, H. Cheng, J. X. Yu, and L. Chen, ``Graph repairing under neighborhood constraints," VLDB J., vol. 26, no. 5, pp. 611-635, Oct. 2017.

[18] A. Bifet, ``Mining big data in real time," Informatica, vol. 37, no. 1, 2013.

[19] Gorton, I. (2020). Foundations of Scalable Data Science. O'Reilly Media. Chapter 1, can be accessed at https://learning.oreilly.com/library/view/foundations-of-scalable/9781098106058/ch01.html#examples_of_system_scale_in_the_early_t.

[20] Lawton, G. (2022, September 9). 7 data quality best practices to Improve Data Performance: TechTarget. Data Management. Retrieved April 6, 2023, from https://www.techtarget.com/searchdatamanagement/tip/Data-quality-best-practices-to-improve-data-performance

[21] A. Q. Mahlawi and S. Sasi, "Structured data extraction from emails," 2017 International Conference on Networks & Advances in Computational Technologies (NetACT), Thiruvananthapuram, India, 2017, pp. 323-328, doi: 10.1109/NETACT.2017.8076789.

[22] Oracle. (n.d.). Determining data sources and ownership. Determining Data Sources and Ownership (Sun Directory Server Enterprise Edition 7.0 Deployment Planning Guide). Retrieved April 9, 2023, from https://docs.oracle.com/cd/E19424-01/820-4806/fpdep/index.html

[23] Schott, M. (2022, November 30). What is data ownership and why is it important? Retrieved April 9, 2023, from https://www.y42.com/blog/data-ownership/

[24] What is a data source? definitions and examples. Talend. (n.d.). Retrieved April 9, 2023, from https://www.talend.com/resources/data-source/

[25] Eteng, O. (2023, January 27). What is data extraction ? everything you need to know. Hevo. Retrieved January 9, 2023, from https://hevodata.com/learn/data-extraction/

[26] A. Imawan, F. K. Putri, S. An, H. -Y. Jeong and J. Kwon, "Scalable extraction of timeline information from road traffic data using MapReduce," 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA), Paris, France, 2015, pp. 1-8, doi: 10.1109/DSAA.2015.7344850.

[27] AirOps Team. (2023, April 2). Data ingestion 101: What it is & how to choose the right tool. AirOps. Retrieved April 9, 2023, from https://www.airops.com/blog/data-ingestion-101

[28] Oracle. (n.d.). What is a data warehouse? Oracle. Retrieved April 21, 2023, from https://www.oracle.com/database/what-is-a-data-warehouse/

[29] M. Kashfi and A. Hajmoosaei, "Optimal Distributed Data Warehouse System Architecture," 2014 IEEE Fourth International Conference on Big Data and Cloud Computing, Sydney, NSW, Australia, 2014, pp. 110-115, doi: 10.1109/BDCloud.2014.52.

[30] Mitruś, P. (n.d.). Data Lake Architecture: How to create a well designed Data Lake. Data Lake Architecture: How to create a well Designed Data Lake. Retrieved April 22, 2023, from https://lingarogroup.com/blog/data-lake-architecture

[31] Singman, P. (2021, November 23). The guide to data versioning. lakeFS. Retrieved April 23, 2023, from https://lakefs.io/blog/data-versioning/

[32] Panian, Z. (2010). Some practical experiences in data governance. World Academy of Science, Engineering and Technology, 62(1), 939-946.

[33] Scherbaum, J., Novotny, M., & Vayda, O. (2018, January). Spline: Spark lineage, not only for the banking industry. In 2018 IEEE International Conference on Big Data and Smart Computing (BigComp) (pp. 495-498). IEEE.

[34] Aikoh, K., Isoda, Y., & Sugimoto, K. (2020, October). Data Profiling Method for Metadata Management. In 2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA) (pp. 779-780). IEEE.

APPENDIX

Appendix A: Datasets & Data Sources Used in the Study

Transaction Data:

This dataset comprehensive sales transaction data for one year obtained from an e-commerce shop. The dataset encompasses essential transaction details, including transaction number, date, product number, product name, price, quantity, customer number, and country of origin. Notably, canceled transactions are denoted by the inclusion of the letter "C" in the transaction number code. Moreover, the data in the dataset is captured daily, with new transactions recorded and added to the dataset at the end of each day. This frequent capture of data ensures that the dataset is updated regularly with the latest sales data, allowing for timely analysis and decision-making based on up-to-date information. The daily capture frequency also helps in identifying any trends or patterns in sales data in near real-time, facilitating agile and informed decision-making. Furthermore, the dataset exhibits a high level of data completeness, with minimal missing values in key columns such as TransactionNo, Date, ProductNo, Product, Price, Quantity, CustomerNo, and Country. This indicates that the dataset contains comprehensive information for most of the transactions captured, providing a reliable and robust foundation for data analysis. However, it is worth noting that there may be some missing values in the Quantity column for canceled transactions, which can be addressed during the data cleaning process to ensure a more complete and accurate dataset. Subsequently, data accuracy is a critical aspect of data quality, and the captured data is thoroughly checked for accuracy during the data capture process. Validation checks are in place to ensure that the TransactionNo, ProductNo, and CustomerNo columns contain unique and valid values, eliminating any duplication or inconsistency in the data. Additionally, the Price and Quantity columns are validated to ensure that they contain valid numeric values and are consistent with the nature of the sales transactions. This meticulous validation process ensures that the data in the dataset is reliable and accurate, minimizing the risk of erroneous or misleading analysis results. Finally, the data in the dataset is captured with minimal latency, as new transactions are recorded and added to the dataset daily. This means that the dataset is updated regularly and reflects the most recent sales data available, allowing for near real-time analysis and decision-making. The minimal latency in data capture ensures that the dataset remains current and relevant, enabling timely identification of sales trends, patterns, and anomalies for effective business decision-making.

Table 10: Transaction Data

Column Name	Description	Collection Method	Data Quality
TransactionID	Categorical column representing a unique six-digit transaction number. "C" in the code indicates a cancellation.	Captured from online sales platform and POS system	A transaction number is a primary key, no missing values are expected
Date	Numeric column representing the date when each transaction was generated.	Captured from an online sales platform or POS system	No missing values expected
ProductID	Categorical column representing a unique five or six-digit product number used to identify a specific product.	Captured from online sales platform, POS system, or sales receipts/invoices	Some missing values are expected, as not all products may have a product number
ProductName	Categorical column representing the product/item name.	Captured from online sales platform, POS system, or sales receipts/invoices	Some missing values are expected, as not all products may have a product name
Quantity	Numeric column representing the quantity of each product per transaction. Negative values related to canceled transactions.	Captured from online sales platform, POS system, or sales receipts/invoices	No missing values expected

Column Name	Description	Collection Method	Data Quality
Price	Numeric column representing the price of each product per unit in Euros (€).	Captured from online sales platform, POS system, or sales receipts/invoices	No missing values expected
CustomerID	Categorical column representing a unique five-digit customer number.	Captured from online sales platforms, POS systems, or customer data collection methods such as customer registration or surveys	Some missing values are expected, as not all customers may have a customer number
Country	Categorical column representing the name of the country where the customer resides.	Captured from online sales platforms, POS systems, or customer data collection methods such as customer registration or surveys	Some missing values are expected, as not all customers may have a country of residence

[a] This table shows columns that are in this dataset, a brief description of what they are about, as well as, how this data is collected and the quality of it.

This table presents comprehensive sales transaction data for one year obtained from an e-commerce shop. The dataset encompasses essential transaction details, including transaction number, date, product number, product name, price, quantity, customer number, and country of origin. Notably, canceled transactions are denoted by the inclusion of the letter "C" in the transaction number code. The data contained in this table is obtained through diverse data sources, encompassing the online sales platform, point-of-sale (POS) system, sales receipts/invoices, and customer data collection techniques such as customer registration or surveys. During the sales process or through recording from sales receipts/invoices, transaction details, including transaction number, date, product number, product name, price, quantity, customer number, and country, are captured and documented. Following data collection, the acquired data is stored in a digital database or spreadsheet, depending on the data management system employed by the shop. To enable efficient data analysis and retrieval, the data may be organized in a structured format with appropriate data types assigned to each column. The dataset encompasses a significant volume of 500K rows, signifying 500,000 sales transactions recorded over the course of one year from the UK-based e-commerce shop. However, the actual size of the dataset may vary depending on the level of data captured for each transaction and the level of detail recorded in the sales transaction data. It should be noted that the data in this table may contain missing values, as not all products may have a product number or name, customer records may lack customer numbers or country of residence, and some transactions may be canceled, denoted by the "C" in the transaction number. Notably, the transaction number is expected to serve as a primary key and, thus, should not contain any missing values. As such, data cleanings and validation techniques, such as data imputation or removal of incomplete records, may be necessary to ensure data quality and accuracy in subsequent data analysis or modeling tasks. Furthermore, implementing data validation checks during data collection can help minimize missing values and ensure data completeness.

Web-Analytics Data

The dataset being described is a web analytics dataset obtained from an e-commerce store, namely the one owned by the abovementioned company. The data in this dataset is collected from Google Analytics 360, a popular web analytics tool that provides comprehensive insights into website performance and user behavior. This dataset contains valuable information about website visitors, their interactions with the website, and transactional data related to purchases made on the website. The dataset under consideration is representative of a typical e-commerce website, containing valuable information related to various aspects of website performance. This includes comprehensive traffic source data, providing insights into the origins of website visitors, such as organic traffic from search engines, paid search traffic from online advertising campaigns, and display traffic from banner ads or other display media. Additionally, the dataset includes content data, which sheds light on user behavior on the website, including the URLs of pages visited, how visitors interact with content and other relevant

metrics. Furthermore, the dataset encompasses crucial transactional data, which provides detailed information about the transactions that occur on the Google Merchandise Store website. This includes data related to purchases made by visitors, such as product details, transaction amounts, payment methods, and other transactional attributes. This transactional data can provide valuable insights into customer behavior, purchase patterns, and revenue generation, aiding in understanding the effectiveness of the e-commerce website's sales performance. By analyzing and leveraging this dataset, various data-driven strategies can be formulated to optimize the website's performance. For instance, insights gained from traffic source data can be used to refine marketing strategies and allocate resources effectively to drive traffic to the website. Content data can provide valuable feedback on website usability, user engagement, and content performance, leading to website optimizations and improvements. Transactional data can be used to analyze sales patterns, identify popular products, understand customer preferences, and refine pricing and product strategies. The columns in the dataset are:

- **TimePeriod**: the period for which the data is recorded, that is daily, weekly, or monthly time intervals.
- **Page**: the specific pages on the website that are being analyzed, that is the URLs or titles of the web pages.
- **Title**: the title of the web pages being analyzed, which could provide additional context about the content of the pages.
- **Visits**: the total number of visits or sessions to the website during the specified period. A visit or session refers to a single user engaging with the website within a given time frame.
- **UniqueVisits**: the number of unique visitors to the website during the specified period. Unique visitors represent the number of individual users who have visited the website, regardless of the number of visits or sessions they have made.
- **Views**: the total number of page views or the number of times the pages on the website have been loaded or viewed by visitors during the specified period.
- **AverageTimeViewedSeconds**: the average time, in seconds, that visitors have spent viewing the pages on the website during the specified period.
- **ConversionRate**: the conversion rate, which is the percentage of visitors who have completed a specific goal or conversion action on the website, such as making a purchase or filling out a form.
- **Transactions**: the total number of transactions or purchases made on the website during the specified period.
- **Revenue**: the total revenue or sales generated from the transactions on the website during the specified period.
- **QuantitySold**: the total quantity of products or items sold on the website during the specified period, providing information about the volume of sales.

Additionally, this dataset can be utilized for data-driven decision-making and strategic planning across multiple teams within an e-commerce organization, including marketing, sales, product management, and business intelligence. For instance, the marketing team can use this data to identify effective marketing channels, optimize advertising campaigns, and tailor marketing messages based on user behavior data. The sales team can analyze transactional data to understand sales trends, customer preferences, and opportunities for upselling or cross-selling. The product management team can leverage the data to identify popular products, optimize pricing, and make data-driven decisions about product development and inventory management. The business intelligence team can utilize this dataset to generate insights, create reports, and inform strategic decision-making at the organizational level. In summary, this dataset provides a rich and diverse set of data points that are essential for understanding the performance of an e-commerce website. It encompasses traffic source data, content data, and transactional data, which can be analyzed to extract meaningful insights, optimize website performance, and inform data-driven strategies across various teams within an e-commerce organization.

Social Media Data - Twitter

The social media data in this dataset is collected using the Twitter API, which allows for the retrieval of publicly available tweets based on certain search criteria or user profiles. The Twitter API provides access to real-time and historical tweets and allows for the collection of large volumes of data for analysis purposes. However, there are several data quality considerations to take into account. First, the dataset may contain incomplete tweets, with truncated or missing information such as hashtags, mentions, or URLs. Second, the accuracy of tweet content, including potential misspellings, grammatical errors, or misinformation, should be validated. Third, noise in the form of irrelevant tweets not related to customer churn or the product/service being analyzed should be filtered out. Fourth, data consistency should be ensured by cleaning and transforming the data to a standardized format. Fifth, data integrity should be checked by identifying and handling duplicate tweets, removing potential spam or bot-generated tweets, and ensuring the data is authentic. Lastly, privacy and ethical considerations, such as adhering to data protection regulations, anonymizing PII, and following ethical guidelines, should be taken into account when dealing with social media data.

The social media dataset under consideration is intended for analyzing customer churn, and it comprises data extracted from tweets posted by customers. Such data can provide valuable insights into customer behavior and sentiment towards a particular product or service. The dataset is expected to contain multiple columns, including but not limited to the following:

- **ID**: This column is likely to contain a unique identifier or ID assigned to each tweet in the dataset. Such identifiers can be used for reference and analysis purposes, allowing for the identification and tracking of individual tweets throughout the analysis.
- **Tweet_Posted_Time**: This column is expected to contain the timestamp or date and time in Coordinated Universal Time (UTC) when each tweet was posted by the customer. This temporal information can be analyzed to understand the timing and frequency of tweets, and how it may correlate with customer churn. For instance, it may be possible to identify patterns in the timing of tweets, such as increased activity during specific periods, which could provide insights into customer behavior and potential triggers for churn.
- **Tweet**: This column is likely to contain the text or content of the tweets posted by customers. The tweets may encompass a wide range of content, including comments, feedback, complaints, or other messages related to the product or service being analyzed. This text data can be subjected to various natural languages processing techniques, such as sentiment analysis, keyword extraction, and topic modeling, to derive meaningful insights. For example, sentiment analysis can help identify the sentiment expressed in the tweets (e.g., positive, negative, neutral), which can provide insights into customer sentiment towards the product or service and its potential impact on customer churn.

In addition to these columns, the dataset may also contain other relevant data, such as user profile information, tweet engagement metrics (e.g., retweets, likes), location data, or hashtags used in the tweets. Such additional data can further enrich the analysis and provide additional context for understanding customer churn behavior. However, it is important to note that the quality of the social media dataset can significantly impact the accuracy and reliability of the analysis results. Factors such as data completeness, accuracy, noise, data consistency, data integrity, and privacy and ethical considerations should be carefully considered and addressed to ensure the integrity of the findings obtained from the dataset. For instance, incomplete or inaccurate data may lead to biased or unreliable analysis results, and noise in the form of irrelevant tweets or spam tweets can distort the findings. Therefore, thorough data cleaning, validation, and transformation procedures should be employed to ensure data quality and reliability. Moreover, privacy and ethical considerations should be taken into account, such as adhering to relevant data protection regulations, anonymizing any personally identifiable information (PII), and conducting the analysis in compliance with ethical guidelines and best practices. In conclusion, the social media dataset being analyzed for customer churn is expected to contain various columns, including *ID*, *Tweet_Posted_Time*, and *Tweet*, which can provide valuable insights into customer behavior and sentiment. However, it is crucial to carefully consider and address

data quality, privacy, and ethical considerations to ensure the reliability and validity of the analysis results.

Sales Data

The dataset represents a comprehensive collection of historical sales data from three distinct sources, providing a wide range of insights into consumer behavior and market trends. The data is sourced from a physical store, an online retail shop, and a third-party data provider, covering a period of six years, from 2014 to 2019. With a diverse set of columns including OrderID, CustomerName, Category, SubCategory, City, OrderDate, Region, Sales, Discount, Profit, State, and PurchaseAddress, the dataset offers a holistic view of various aspects of sales transactions. Also, we should keep in my that, columns such as PurchaseAddress, and CustomerName should be anonymized before processing it. This comprehensive dataset offers a wealth of information for sales analytics, market analysis, and predictive data analytics. It can be utilized to gain insights into consumer preferences, market trends, and sales performance, allowing businesses to set targets, forecast future sales, and make data-driven decisions. Whether it's analyzing the sales performance of different product categories, understanding the impact of discounts on sales, identifying top-performing products or underperforming segments, or evaluating customer satisfaction through ratings, this dataset provides a robust foundation for in-depth analysis and actionable insights. The dataset is a valuable resource for businesses seeking to optimize their sales strategies, enhance customer experiences, and drive revenue growth in today's highly competitive market landscape. Description of columns:

- **OrderID**: This column represents the unique identifier for each sales order. It is generated by the respective data sources and serves as a primary key for the dataset. The OrderID is collected through automated systems for online sales and point-of-sale (POS) systems for in-store sales.
- **CustomerName**: This column captures the name of the customer who made the purchase. It is collected through customer registration or account information, and in the case of pharmacy data, it may also include prescription details. The data collection method varies depending on the source, with online sales capturing customer names through online registrations, while in-store sales may require manual data entry or scanning of customer loyalty cards.
- **Category**: This column represents the broad category or type of product sold in the sales transaction. It includes categories such as Electronic accessories, Fashion accessories, Food and beverages, Health and beauty, Home and lifestyle, Sports and travel, and Anatomical Therapeutic Chemical (ATC) Classification System for drugs in the pharmacy data. The category information is typically collected through product codes or descriptions provided by the retailer or pharmacy.
- **SubCategory**: This column provides further granularity within the broader category, capturing the specific sub-category or type of product sold. For example, supermarket data may include sub-categories such as Mobile phones, Laptops, Clothing, Groceries, etc. The sub-category information is collected in a similar manner as the category, through product codes or descriptions provided by the retailer or pharmacy.
- **City**: This column captures the city or location where the sales transaction occurred. It is typically obtained from the billing or shipping address provided by the customer during the purchase process for online sales or through the point-of-sale (POS) system for in-store sales.
- **OrderDate**: This column represents the date of the sales order, capturing the day, month, and year of the transaction. It is collected through the respective data sources' transactional systems, such as online sales platforms or point-of-sale (POS) systems in the case of in-store sales.
- **Region**: This column captures the region or geographic location where the sales transaction occurred. It may include information such as country, state, or province, depending on the data source. The region information is obtained from the billing or shipping address provided by the customer for online sales or through the point-of-sale (POS) system for in-store sales.
- **Sales**: This column represents the total sales amount for the transaction, including any taxes or fees. It is collected as the gross revenue from the sales transaction and may be recorded in different

currencies, depending on the data source. The sales data is captured automatically from the respective data sources' transactional systems.

- **Discount**: This column captures any discount applied to the sales transaction, expressed as a percentage or a fixed amount. It may be provided by the retailer or pharmacy as part of promotional offers or customer-specific discounts. The discount information is collected from the respective data sources' transactional systems or promotional data.
- **Profit**: This column represents the profit or margin earned from the sales transaction, calculated as the difference between the sales amount and the cost of goods sold (COGS). It is typically obtained from the respective data sources' financial systems, which track the cost of goods sold and other relevant expenses.
- **State**: This column captures the state or province where the sales transaction occurred, providing more specific geographic information. It is obtained from the billing or shipping address provided by the customer for online sales or through the point-of-sale (POS) system for in-store sales.
- **Purchase Address**: This column captures the complete address where the sales transaction occurred, including the street address, city, state/province, and postal code. It is obtained from the billing or shipping address provided by the customer for online sales or through the point-of-sale (POS) system for in-store sales.

The dataset is carefully curated to ensure high data quality. However, like any real-world dataset, it may contain some imperfections. Data quality measures include identifying and addressing missing values, inconsistencies, and errors in data entry. For example, missing values in customer names, product categories, sub-categories, or other columns may occur due to incomplete or inaccurate data entry during the data collection process. Data quality checks and data cleaning techniques are applied to minimize such issues and ensure the accuracy and reliability of the dataset. In conclusion, this comprehensive dataset includes historical sales data from multiple sources, capturing various aspects of sales transactions, customer information, and product details. The dataset is collected through automated systems for online sales and point-of-sale (POS) systems for in-store sales and includes data on customer names, product categories, sub-categories, city, order dates, region, sales, discounts, profits, state, and purchase addresses. Data quality measures are implemented to ensure the accuracy and reliability of the dataset for analysis and insights into consumer behavior and market trends.

Product Review Data

The dataset provided appears to be a comprehensive collection of customer reviews for product X on the e-commerce website of Company A. The dataset includes various fields, such as *review title, review content*, *reviewer information* (such as name and location), *rating*, *likes*, *dislikes*, *average rating*, *category*, *crawled date, product description*, *images*, *price*, *URL*, and *whether the purchase was verified*. This dataset is likely utilized for sentiment analysis, product evaluation, and customer feedback analysis by Company A. It can provide valuable insights into overall customer satisfaction with product A, including its features, performance, and areas for improvement. The method of data collection for this dataset was not explicitly mentioned by the company, but it appears to be obtained from customer reviews submitted on the Company A website. It is possible that Company A uses web scraping techniques to collect reviews from their product pages or they have internal tools that make this process easier and more efficient.

In terms of data quality, several observations can be made based on the given entry. Firstly, regarding data completeness, the entry contains various fields, such as *review title, review content, reviewer information, rating, likes, dislikes*, etc. However, some fields are missing or incomplete, such as the "reviewed_at" field, which is empty, and fields like "review_uniq_id" and "crawled_at" that have incomplete values. Moreover, data accuracy of the data cannot be fully verified as there is no information on how the data was collected and processed. Furthermore, data consistency, the data is generally consistent, with fields like "rating" and "likes" containing numeric values and fields like "verified_purchase" containing boolean values. However, there are discrepancies in the format of some

fields, such as "price" which is represented with a currency symbol and may require further processing to extract meaningful numerical values. Next up on the list, is data validity, the dataset contains subjective information such as review content, which may be biased or opinionated. The "verified_purchase" field indicates whether the review was submitted by a verified buyer, but it is not clear how this verification is done and if it is reliable. Finally, data currency: The "crawled_at" field indicates the date and time when the data was crawled, which is mentioned as "02/09/2021, 01:37:30". It should be noted that the dataset may not contain the most up-to-date information as it was crawled at a specific point in time.

It is important to note that the quality and reliability of the dataset provided can significantly impact the accuracy and validity of any analysis or insights derived from it. Therefore, careful consideration of data quality and potential biases should be taken into account when utilizing this dataset for any analysis or decision-making purposes in an academic or research context. Keys:

- **_id**: This column captures the unique identifier for each review entry in the dataset. It is generated by the system and serves as a primary key for data management and retrieval.
- **average_rating**: This column captures the average rating given by customers for a specific product. It is a numerical value representing the average of all the ratings provided by customers. It is obtained from customer reviews and ratings submitted on an online retail platform.
- **category**: This column captures the category of the product being reviewed.
- **crawled_at**: This column captures the timestamp when the review data was crawled or extracted from the online retail platform. It provides information about the time and date when the data was collected for analysis.
- **description**: This column captures the detailed description provided by the reviewer about their experience with the product. It includes information about various aspects of the product. The description is obtained from the review text submitted by customers on the online retail platform.
- **dislikes**: This column captures the number of dislikes received by the review entry. It is a numerical value indicating the count of customers who expressed dissatisfaction with the review. It is obtained from customer interactions, such as dislikes or thumbs-down ratings, on the online retail platform.
- **images**: This column captures the URLs of the images associated with the product being reviewed. It includes multiple URLs pointing to different product images. The URLs are obtained from the image attachments submitted by customers along with their reviews on the online retail platform.
- **likes**: This column captures the number of likes received by the review entry. It is a numerical value indicating the count of customers who expressed satisfaction with the review. It is obtained from customer interactions, such as likes or thumbs-up ratings, on the online retail platform.
- **location**: This column captures the location of the reviewer who provided the review. The location is obtained from the reviewer's profile information on the online retail platform.
- **price**: This column captures the price of the product being reviewed. It is a numerical value with currency information, indicating the cost of the product at the time of the review. The price is obtained from the product information available on the online retail platform.
- **product**: This column captures the name or title of the product being reviewed. The product name is obtained from the product information available on the online retail platform.
- **rating**: This column captures the overall rating given by the reviewer for the product. It is a numerical value ranging from 1 to 5, indicating the reviewer's overall satisfaction level with the product. The rating is obtained from the customer's review and rating submission on the online retail platform.
- **review_title**: This column captures the title or summary of the review provided by the reviewer. It is a concise representation of the main point or sentiment expressed in the review text. The review title is obtained from the review text submitted by customers on the online retail platform.

- **review_uniq_id**: This column captures a unique identifier for each review entry in the dataset. It is generated by the system and serves as a secondary key for data management and retrieval.
- **reviewed_at**: This column captures the timestamp when the review was submitted by the

Appendix B: Figures Relating Configurations and Setting up

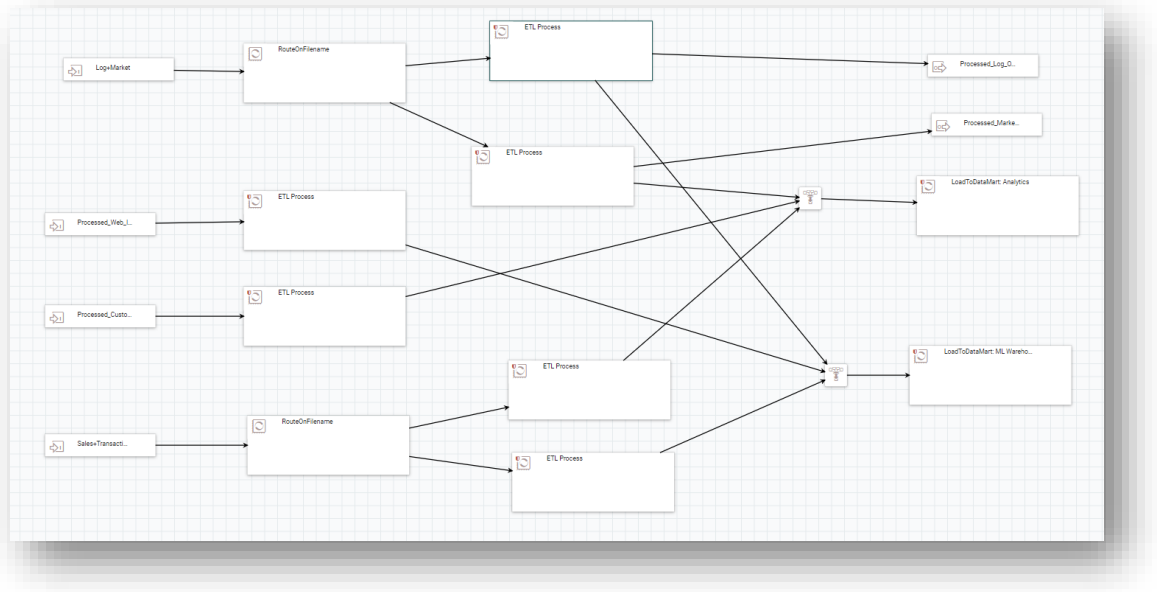

Figure 14: Apache Nifi. Data Science domain of the Pipeline

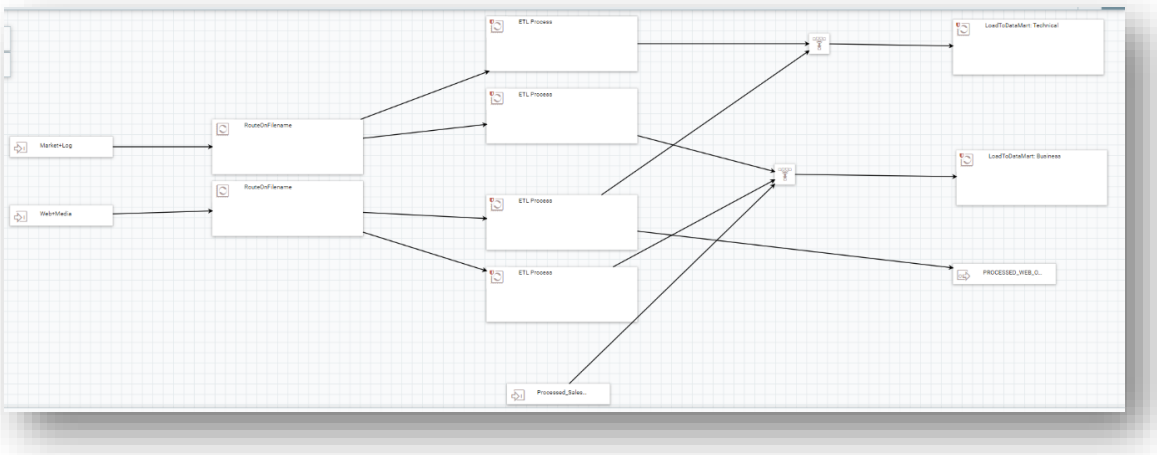

Figure 15: Apache Nifi. Business Intelligence domain of the Pipeline

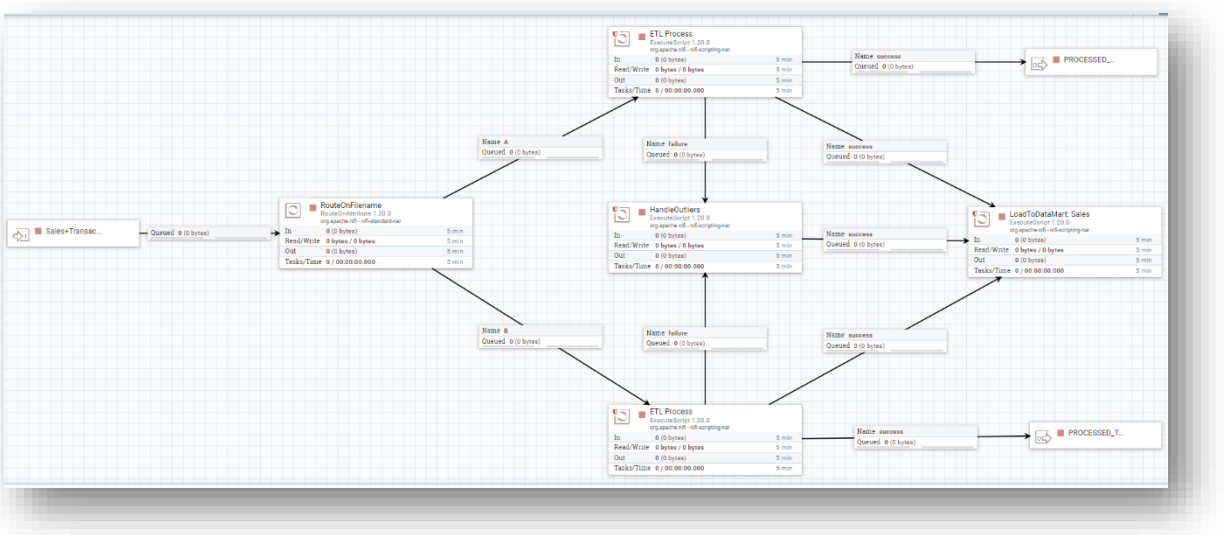

Figure 16: Apache Nifi. Sales domain of the Pipeline

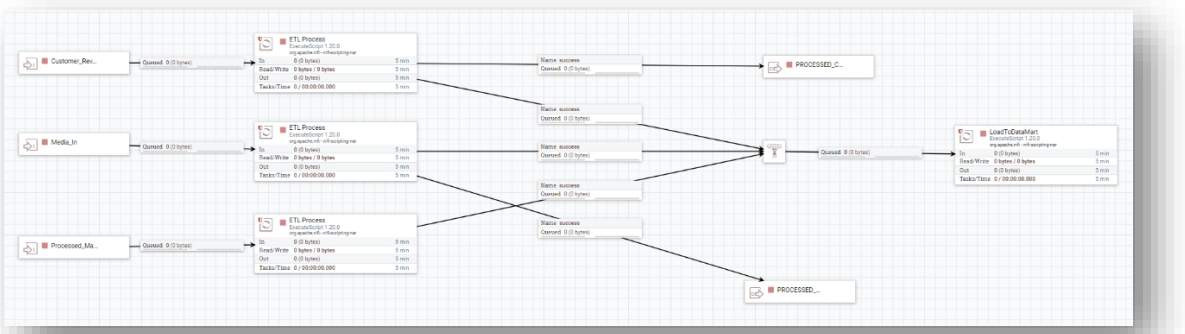

Figure 17: Apache Nifi. Marketing domain of the Pipeline

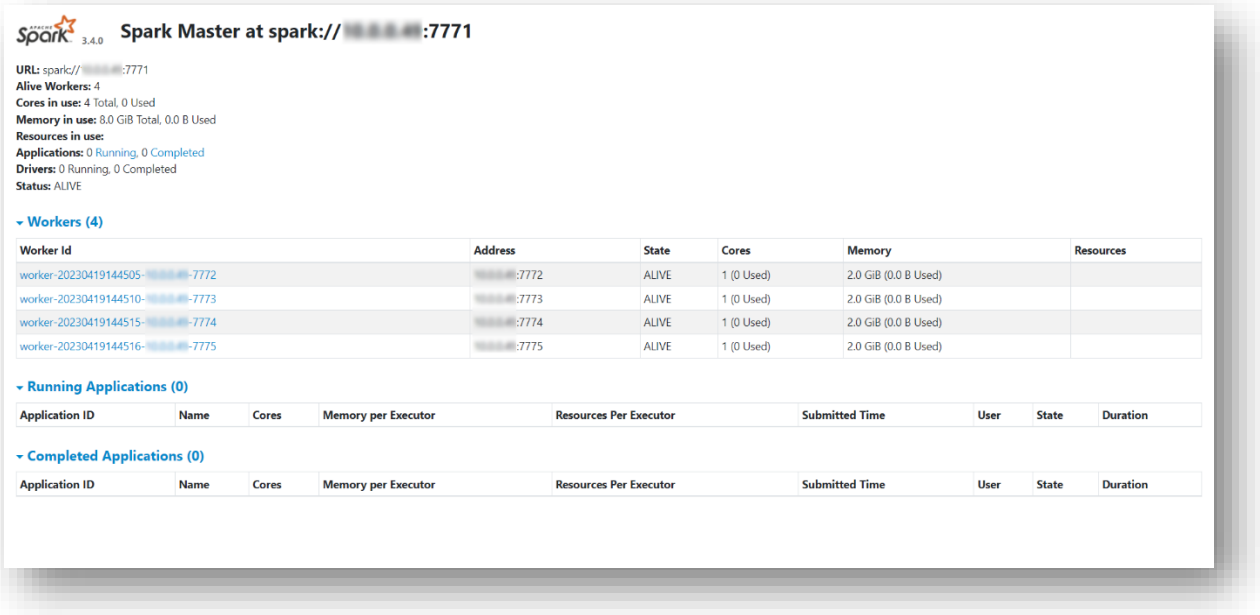

Figure 18: Up and Running Installation of Apache Spark. Current instance: 4 workers with a single processing core each, 8.0 GiB Total Memory with 2.0 GiB per worker.

Appendix C: Figures and Graphs

Benchmark Results			Test 1			Test 2			Test 3			Test 4			Test 5		
			#1	#2	#3	#1	#2	#3	#1	#2	#3	#1	#2	#3	#1	#2	#3
MBytes Read	Original Pipeline	Min	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
		Max	170.61	213.26	113.08	989.98	1,079.90	809.98	3,014.00	3,288.00	2,192.00	35,610.00	29,310.00	26,150.00	-	-	-
		Mean	1.01	1.27	0.63	19.24	23.68	11.36	163.16	190.36	81.58	1,760.00	1,310.00	1,100.00	-	-	-
	Proposed Pipeline	Min	-	-	-	-	-	-	-	-	-	-	-	-			
		Max	44.55	84.36	38.29	524.99	614.99	449.99	2,180.00	2,380.00	1,590.00	33,980.00	32,380.00	27,580.00	32,550.00	27,630.00	26,630.00
		Mean	0.41	0.78	0.31	11.10	12.81	8.06	117.00	136.50	58.50	1,130.00	1,070.00	846.50	2,410.00	1,970.00	1,520.00

Benchmark Results			Test 1			Test 2			Test 3			Test 4			Test 5		
			#1	#2	#3	#1	#2	#3	#1	#2	#3	#1	#2	#3	#1	#2	#3
MBytes Written	Original Pipeline	Min	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
		Max	88.76	55.48	122.21	882.72	962.97	682.22	2,000.00	1,780.00	2,440.00	17,330.00	21,670.00	1,845.00	-	-	-
		Mean	3.98	1.02	2.60	23.18	28.53	13.10	139.52	104.64	209.29	1,430.00	1,930.00	1,560.00	-	-	-
	Proposed Pipeline	Min	-	-	-	-	-	-	-	-	-	-	-	-			
		Max	214.18	115.71	72.28	1,101.96	1,182.96	818.97	4,550.00	3,590.00	3,190.00	25060	27230	20660	37,920.00	31,490.00	37,920.00
		Mean	18.57	9.24	6.19	25.68	32.07	16.82	354.80	236.00	196.00	2260	2550	1680	1,360.00	1,340.00	1,620.00

Figure 19: Table of data collected during the benchmark. It contains data captured from 3 different timestamps during the 5 tests. Shows how many megabytes per 5 minutes can original pipeline and proposed pipeline can read/write.

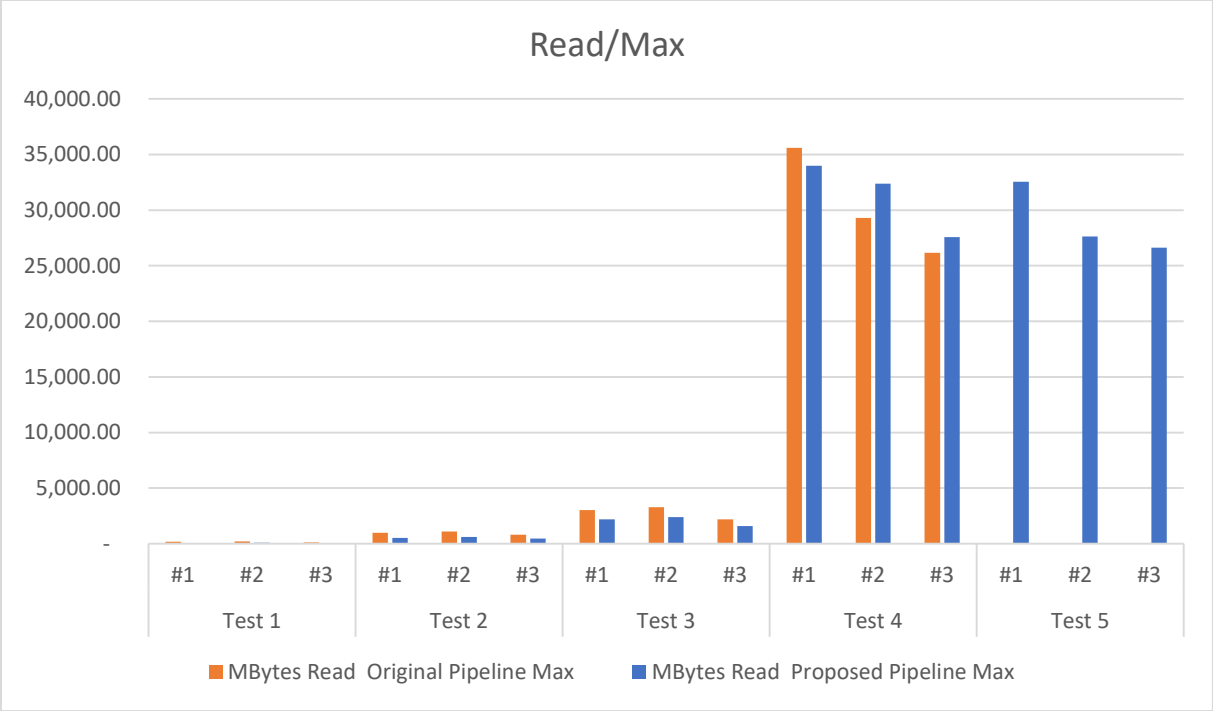

Figure 20: The bar graph shows the maximum size (in megabytes) that was read in 5 minutes by both pipelines.

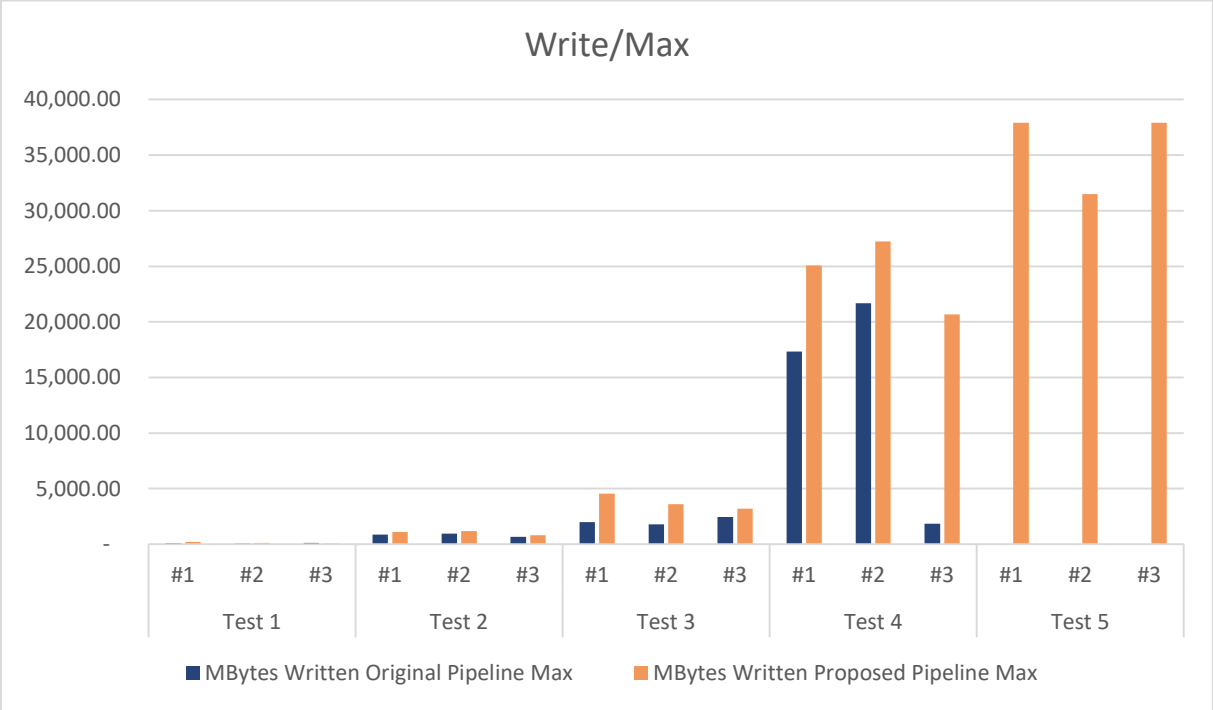

Figure 21: The bar graph shows the maximum size (in megabytes) that was written in 5 minutes by both pipelines.

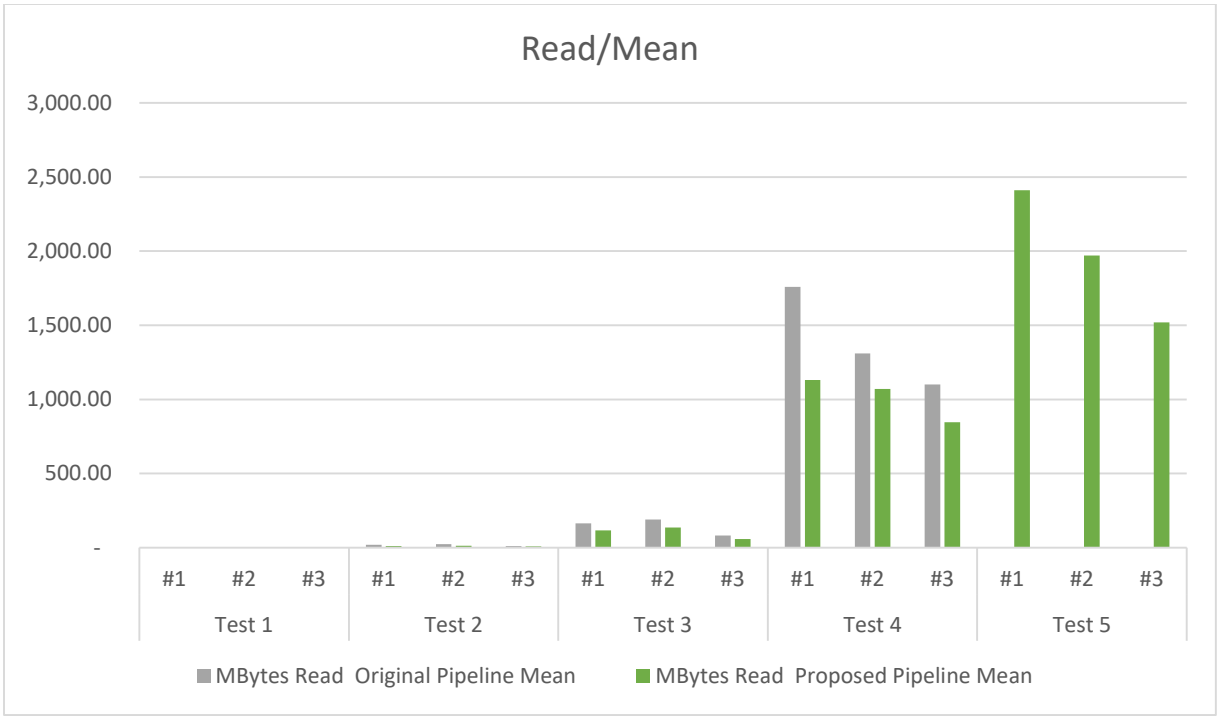

Figure 22: The bar graph shows the mean size (in megabytes) that was read in 5 minutes by both pipelines.

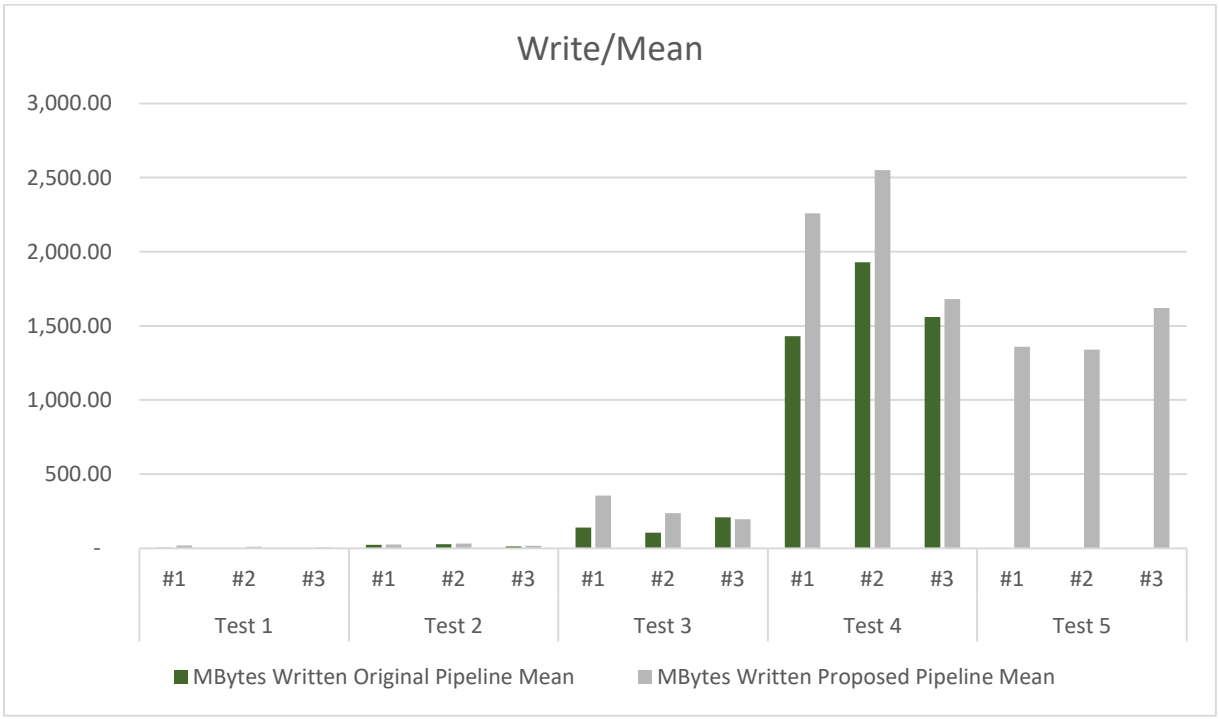

Figure 23: The bar graph shows the mean size (in megabytes) that was written in 5 minutes by both pipelines.

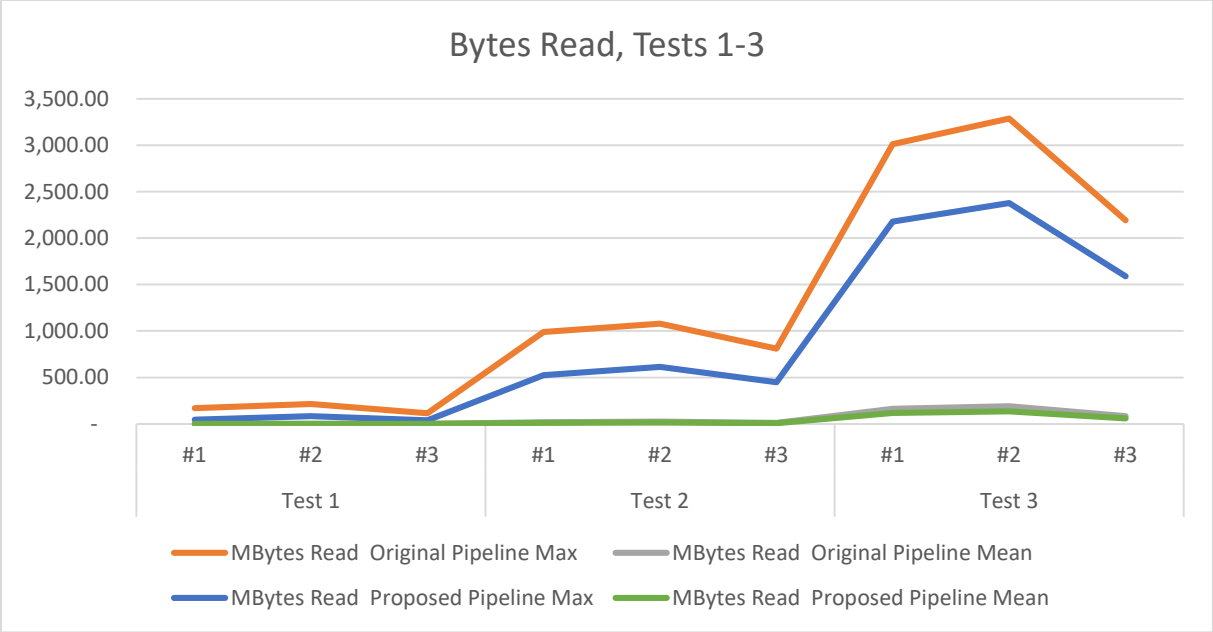

Figure 24: Line chart shows the difference between both the maximum and means of read data's size in 5 minutes throughout Test 1 to 3.

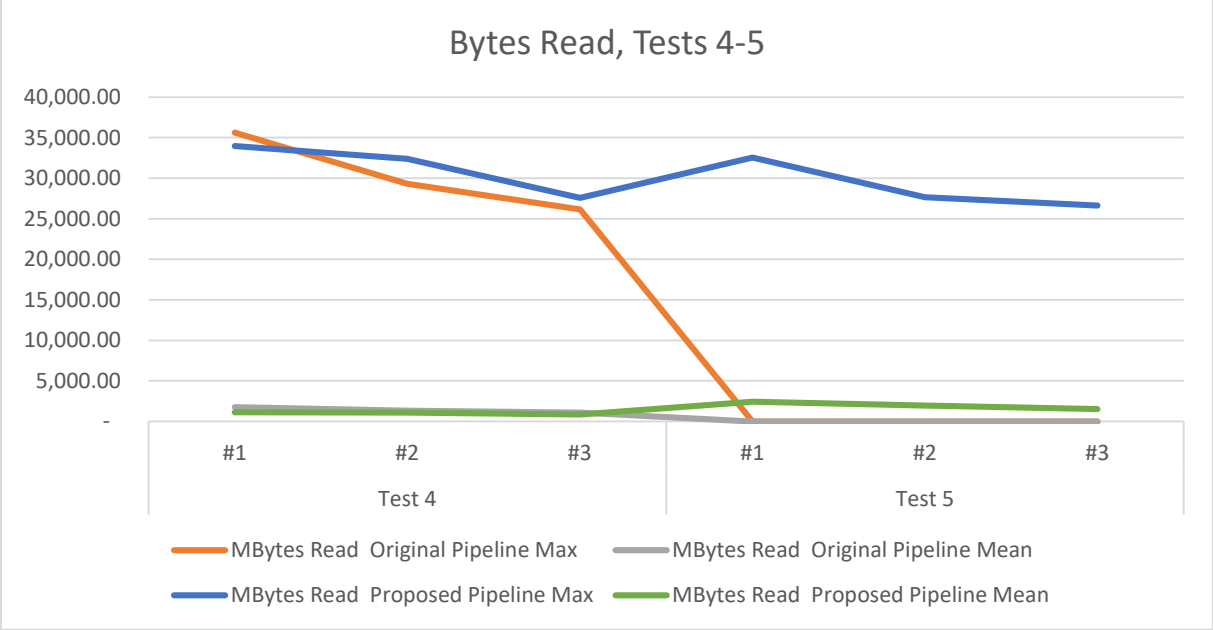

Figure 25: Line chart shows the difference between both the maximum and means of read data's size in 5 minutes throughout Test 4 – 5. Note that the original pipeline did not participate in Test 5.

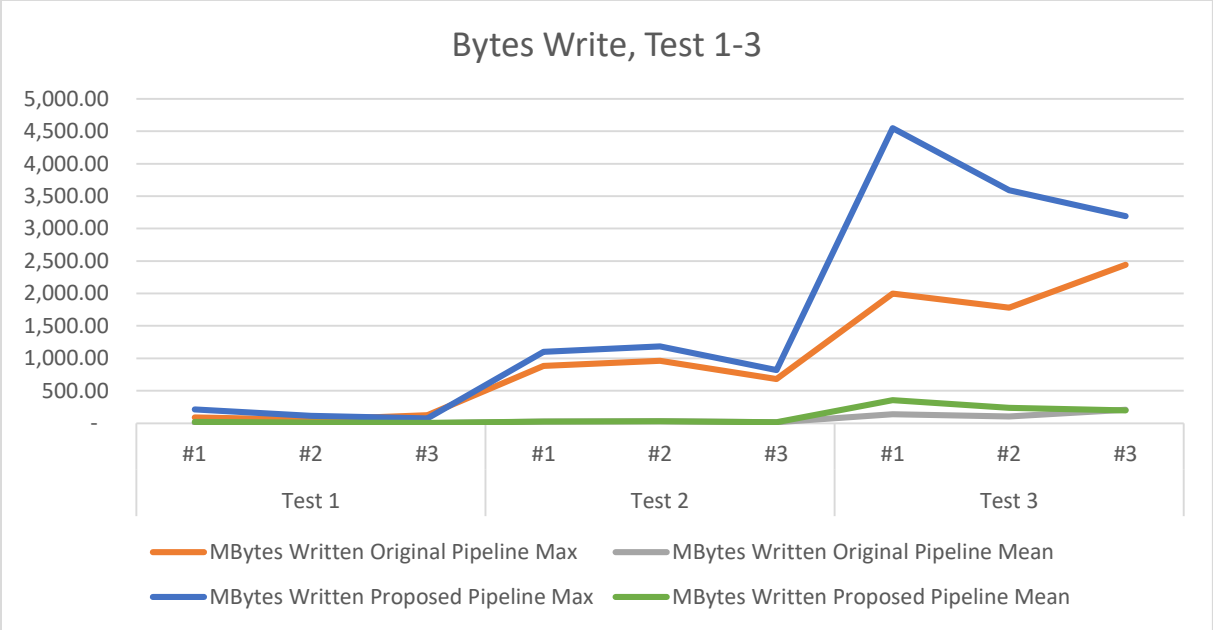

Figure 26: Line chart shows the difference between both the maximum and mean of written data's size in 5 minutes throughout Test 1 to 3.

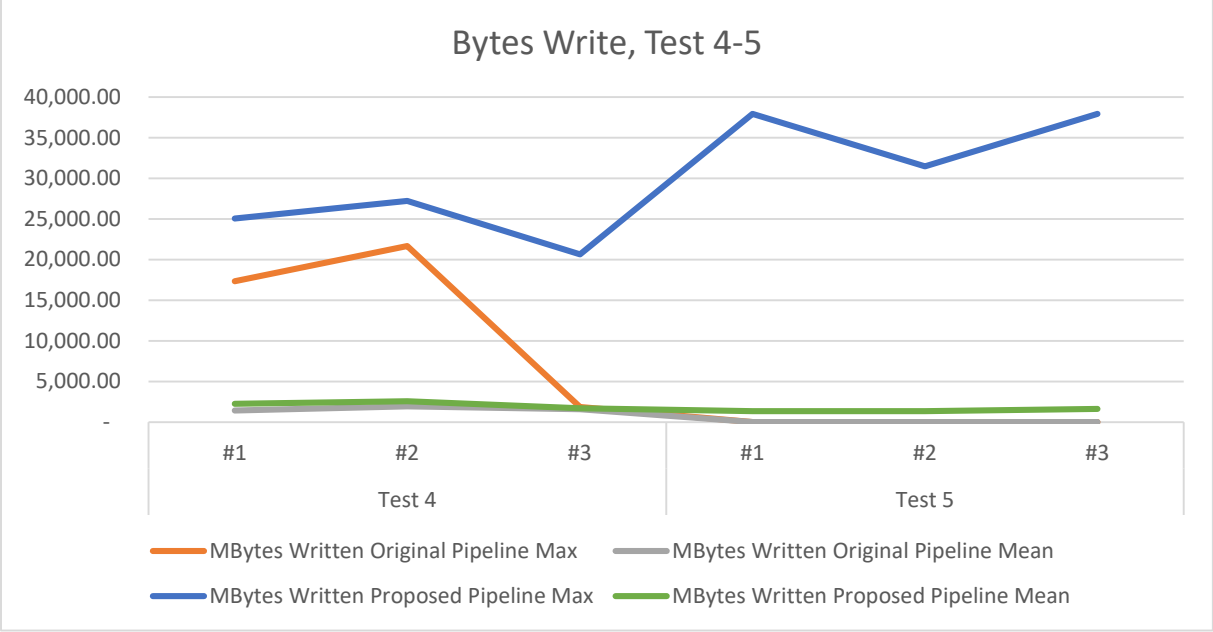

Figure 27: Line chart shows the difference between both the maximum and mean of written data's size in 5 minutes throughout Test 4 – 5. Note that the original pipeline did not participate in Test 5.