***Abstract*—**Data ingestion constitutes a crucial step in processing and analyzing extensive volumes of data. The advancement in recent years has seen the emergence of a plethora of tools and technologies designed to facilitate the construction of efficient and scalable data ingestion pipelines. In this research, we employed Apache NiFi and Apache Kafka to devise a data ingestion pipeline, transferring data from its origin to a designated output destination like a database or data lake, and a data management system using lakeFS. Our constructed data pipeline employed Apache NiFi for orchestrating the data flow, while Apache Kafka was used to address data ingestion and delivery. MongoDB was our chosen output destination, favored for its competence in handling unstructured data and simplicity of use. LakeFS, renowned for its version control capabilities over objects, was used to manage the ingested data. To ascertain the performance of our data pipeline, we conducted a series of experiments, collecting a range of metrics encompassing message throughput, latency, and resource utilization. The results demonstrated that our data pipeline, leveraging Apache NiFi and Apache Kafka, was not only efficient but also exhibited scalability and reliability in managing substantial volumes of data.

***Index Terms*—Big Data, Data Pipeline, Data Ingestion, Data Lake, Real Time,**

## I. INTRODUCTION

### *A. Motivation*

Traditional batch analytic approaches will wait till the data received achieved a certain amount which can cause the data to be outdated. Real time ingestion is aiming for processing the data at the time it is obtained to make the process more effective and efficient for storage. In the big data era, the amount of information that needs to be processed now is enormous, finding a way to process that data is one of the major topics. In current study, real time data analysis is expensive and complex, that is why we want to find a less costly way to ingest the data and build a pipeline to make it less complex. And a modern solution to storing big data is a data lake, however, the lack of a data management system can turn a data lake into a data swamp which is why we want to explore data management solutions.

### *B. Problem Description*

To utilize real-time data within this project, we first need to capture and store it in a database. Specifically for this project, we have chosen Microsoft MongoDB as our storage solution. Subsequently, we will execute a data ingestion process, refining the data for later utilization. Once the data has been processed, we will deploy machine learning models to analyze the data with the aim of addressing specific problems. A considerable portion of this project involves the implementation of data ingestion models. This includes leveraging tools such as Nifi with Kafka, Apache Spark with Nifi, Apache Storm, and Flume. Post ingestion, we will embark on data analytics, applying techniques such as machine learning on our selected dataset.
Following the ingestion process, we will undertake data management, meticulously tracking the ingested data to ensure both effectiveness and efficiency. To appraise the success of our ingestion process, we propose to evaluate the memory usage and time spent for various ingestion tools, subsequently providing a comparison analysis. Regarding data analytics, we plan to divide the data into training and testing subsets to evaluate the accuracy of our models. Despite these diverse tasks, the primary focus of the project will remain on the data ingestion pipeline.

### *C. Key Contributions*

A pivotal contribution of this project centers on optimizing the data ingestion pipeline to guarantee efficient processing and storage of real-time data. This endeavor entails exploring a variety of data ingestion tools including Nifi with Kafka, Apache Spark with Nifi, Apache Storm, and Flume. The selection of the most appropriate tool will hinge on several factors such as speed, memory usage, and scalability. Additionally, we aim to highlight beneficial features that aid in managing ingested data, employing tools such as lakeFS. By optimizing this crucial aspect of data management, we aim to significantly enhance the overall efficiency and effectiveness.

## II. LITERATURE REVIEW

### *A. Big data real time ingestion and machine learning*

The paper addresses the presentation of a live streaming data ingestion and processing mechanism, and big data machine learning modules. The paper has expressed the preliminaries of the ingestion including Processing time series data, Data Processing Patterns, Windowing, Watermarking, Spark D-Streams. For the streaming data and processing the paper has experimented on two ingestion frameworks - Kafka and Flume. Under the two frameworks the paper has used 4 different tools for comparison: Apache Kafka, Apache Spark and Apache Storm, Apache Cassandra, Flume.
The data set the paper is using os clickstream data which is the recording of parts of the screen a user clicks on while web browsing. To capture the data the paper expressed two approaches. First approach is to collect the web browser log files and then push them into HDFS. Second approach is that the event is generated on the client side and delivered to the separate back-end service to handle the event processing and logging while still in transit without the need for storing. As the second approach has more advantage in efficacy, therefore the paper has adapted the second approach.
By comparing Kafka and Flume the paper found that Flume would be easier to implement being a configuration based contrary to Kafka's Programming-based integration, and data transformation can be processed through Flume interceptors. While Kafka needs to integrate with other data processing frameworks. The paper also tried using both Kafka and Flume. For analysis of the data, the paper used two different machine learning techniques, Streaming Regression and Streaming K-Means Clustering. For future work, the other architecture can be extended for processing approaches. And for now, the

stream layer serves only low-latency queries which can be changed.

### *B. Recent Trends in Big Data Ingestion Tools: A Study*

This paper has presented a comparative study of big data ingestion tools. Also discussed the correct implementation in research areas, and performance of the tool on different tasks. The tools that the paper has discussed are the following: Apache Kafka, Apache NIFI, Amazon Kinesis, Apache Flink, Apache Storm, and Apache Gobblin. Apache Kafka is basically an actual time data injection tool. It performs better as a high throughput distributed messaging system. If the data has no requirement of computation or cleaning and can be directly consumed, then Apache Kafka can be used. Apache NIFI automates the movement of data between varied data sources as well as data lake systems, making ingestion fast, easy, and secure. Amazon Kinesis is a tool hosted by AWS(Amazon) for real-time big data processing. This tool makes accumulating, transforming, and analyzing real-time streaming data are quite easy. Apache Flink is a distributed processing engine or framework for stateful computations over infinite or finite data streams. Apache Storm process task on parallelism principle, also due to its being open source it can be easily operated. Apache Gobblin is a data ingestion tool for fetching, processing, computing, and loading a high volume of data from enormous data sources. The paper also shows the performance of each tool which gives a better understanding and provides a guideline to the other researcher.

### *C. Real-time social media analytics with deep transformer language models: a big data approach*

This paper has built a framework for ingestion, analysis and storage for distributed, fault-tolerant to real-time big data. The paper also reviewed other research on Transfer with Pre-Trained Language Models and Real-Time Stream Analytics Frameworks. To prepare the framework for data ingestion there are three fundamental steps a) data ingestion, b) data analysis and c) data storage. The paper has added data handling and analytic layers on the framework. Data handling manages ingested data, providing greater backlog tolerance in cases of temporary spikes in data ingestion. The analytic layer will let users visualize, actionable insights.
The paper gives a case study to support the framework that has been proposed. Out of all the tools the paper has chosen the Apache NiFi since it can reduce development time and can support wider variety of data also it is monitoring through web GUI provides access to real-time data streams and quicker debugging. For the data handler layer Apache Kafka has been used since it can deal with all types of streaming data. Kafka uses queues where the data are stored until consumed by the analysis component or a specific time or size limit is reached. This can provide backlog tolerance to analytic pipelines. For the storage layer Apache Hadoop provides excellent horizontal scalability with strong fault tolerance. The data analytic pipeline consists of three components 1) Pre-processing 2) tokenizer and 3) inference. Lastly this pipeline has a data analysis layer, where the analysis insights are put in the storage server.

### *D. KAFKA: The modern platform for data management and analysis in big data domain*

This paper has given a comprehensive review of Apache Kafka and Apache Flume, and how to develop applications using both tools. For the data management layer, the paper has shown three components. First is Apache Zookeeper which is a distributed application that enables synchronization across a cluster. Then there is Apache Flume which is collecting and aggregating and moving huge amounts of data like log data efficiently and effectively. Lastly Apache is used to handle ingestion transactions and real time data ingestion. The paper also shows the compression of both tools that states the Flume is better when needing to log data inside a central storage, extract data from multiple servers to Hadopp, and importing large volumes of data events. While Kafka is better at g real-time pipelines of data stream, excellent guarantee of order as compared to traditional or other systems for messaging, and it has the property of scale processing and supports any number of subscribers.
Overall, the paper shows that Kafka can be used when needed as a highly reliable messaging system to connect many multiple systems, and the pull system is also an advantage of Kafka. The future work of this paper can be extended to a more detailed comparison between Kafka and Flume and other applications.

### *E. A Study of Apache Kafka in Big Data Stream Processing*

This paper presents a basic overview of Kafka and examples with experimental results. Kafka is using pub-sub as the messaging system, and it can handle a large volume of data which can allow the message to be sent at the endpoint. The paper describes the benefit of Kafka using five points: scalability - Kafka scales easily without down time, high-volume - it can work with high - volume of data, reliability - high fault tolerant, data Transformations - easy to reinvests new data and low latency. The paper shows the framework of the Kafka which contains topics, producers, connector, consumers, steam power, zookeeper, and broker.

### *F. Real Time Streaming Data Storage and Processing using Storm and Analytics with Hive*

This paper discussed a way of processing big data using Apache storm and hive as the data warehouse for later analysis. The system of producers, brokers and consumers are called the logistic group soi that one copy of the message will be available at Kafka broker, and the consumer is only getting one copy. The Data processing the paper is using is storm since it is good at machine learning and can process million records per second on a single node. For the data analysis the paper used Hive which is a Data warehouse system for Hadoop. The benefit of using Hive is it does not need code written but SQL syntax which is easier for non-developers. The paper has stated the best ecosystem in Hadoop is Apache Storms and the best partner to the Apache Storm is Apache Kafka, the system that has been created is reliable. For the future work for this paper might be a use case where the system has been implemented.

### *G. Big data stream computing in healthcare real-time analytics*

This paper presented a used case for big data analysis – healthcare applications. The data that has been collected in this paper is Electronic Healthcare Records (EHRs), Biomedical imaging, Sensing data, Biomedical signals, Genomic data, Clinical text, and social network.
For the messaging system the paper has used Kafka due to the fast, scalable, and durable nature of the Kafka System. It can handle hundreds of megabytes of reads and writes per second from hundreds of clients. For brokers and consumers to state information and track messages ZooKeepers are used. ZooKeeper is from Apache Storm, and it is designed to be scalable, reliable, resilient, extensible, efficient, and easy to set up and maintain. And Hadoop is providing a set of general primitives for batch computing which data processing is based on.

### *H. Learning to Reliably Deliver Streaming Data with Apache Kafka*

This paper delivered some reliable method for using Apache Kafka, and evaluation model. The paper used the basic Kafka Architecture as the pipeline. All the streaming data will be processed in Kafka Producer and be sent to Kafka as messages. In Kafka all the messages will be clustered and be stored in multiple partitions. Stream processors can then subscribe to a topic via Kafka. To improve the efficiency and accuracy the prediction model that the paper used is ANN since those can be changed by adjusting the structure of the ANN. Changing the configuration parameter shows ideal guaranteed reliability in complex and changing scenarios. The paper shows some takeaways of the Kafka: message should be larger than 300 bytes to prevent message loss, overloaded producers may lose message, batching message before sending can reduce the chance of message loss. The future work of this paper can be using different strategies in Kafka and comparing the performance.

### *I. The Development of Real-time Large Data Processing Platform Based On Reactive Micro-Service Architecture*

This paper has talked about using Kafka as a real time processing control system and big data processing platform. The paper chose Kafka as the streaming because of the high throughout, low latency, and guaranteed reliability when it handles real time data. The paper used Spark as the platform and control system to simplify the development of collecting data and transfer it to Kafka. All the data that has been collected will be sent to Spring Cloud Stream where it uses Spring Boot for configuration, and it will be sent to different Kafka topics. Lastly, the big data framework Spark can read the data from Kafka to process and analyze.

### *J. Performance Evaluation of Apache Kafka – A Modern Platform for Real Time Data Streaming*

This paper has analyzed the key indicators of the Apache Kafka which can help with designing Kafka. For Kafka it is a pub-sub structure. The producer will collect the data and push it to the Kafka Broker. Within the Kafka 'Topic' is used as a storage interface which the consumer is subscribed to. The Producer, Consumer and the Processor in Kafka can execute in parallel. In the test rounds the CPU spike when the data volume increases, partitions increase and the start of producer or consumer process.
This paper helps with determination of the configuration considering the hardware capacity to avoid issues, increase scalability and be more tolerant. The future work could be discussing the configuration with performance with low data lost.

### *K. A Big Data Lake for Multilevel Streaming Analytics*

Data management is challenged by the high volume, velocity, and variety of data in the modern big data world. This paper looks to find a solution in data lakes to effectively handle and analyze data. In the proposed data lake: All data is stored in the data lake or staging repository such as HDFS in HDP. Data can be ingested in various ways and tools like NiFi and Kafka can be used for this purpose. Sqoop is used for transferring bulk data between Hadoop and structured datastores. The ingested content can be analyzed using big data and analytics tools such as Spark. HBase and Hive are used for processing SQL queries in a Hadoop cluster. Hive can be used to insert data into HBase and join data from different sources. Spark is used for high-performance analytics of both batch and streaming data. Analytics results can be visualized through display tools like Zeppelin. The architecture allows for transporting data from a traditional database to Hadoop components to address limitations of traditional computing. The dataset used for this implementation included an SQL database for a car trading firm and Twitter data, which were used to demonstrate an analytical use case for car sales and stocks. The SQL database was a MySQL database with various tables, and the Twitter data consisted of mentions of specific entities related to the company and its car brands. The paper describes the utility of implementing a data lake as a unified data management and analytics platform and provides screenshots to demonstrate major steps of data ingestion and higher-level analytics. The future work involves defining metadata when importing relational data into HDFS using Sqoop and storing the original data into different NoSQL databases to compare computation time.

### *L. From Data Warehouse to Lakehouse: A Comparative Review*

In this big data era, data warehouses cannot efficiently extract, process, integrate and store the high volume, velocity, and variety of data; Data lakes can turn into data swamps due to the delay in processing the raw data caused by multiple factors such as changing data content, inconsistencies and errors in the data, and the need to repeatedly extract metadata. The paper focuses on combining the advantages of data warehouses, data lakes, and current lakehouse systems and proposes a new data lakehouse architecture. An efficient lakehouse should have: A data storage that is accessible, flexible, and cost-effective among others Data ingestion that is fast with knowledgeable extraction and being cost-effective Metadata management with knowledge graphs to include information about linked data and enhance the lakehouse's analytical capabilities Data transformation to clean, preprocess, and summarize before loading into memory, data access with advanced knowledge extraction methods with linked data exploration. For implementation, the paper suggests using Snowflake or Amazon S3 for data storage, an open-source ontology-based tool for adaptive metadata management portfolio such as Egeria, Apache Flink and Apache NIFI for ingestion. The architecture was tested using the Parquet file system. The lakehouse achieved a 50% performance improvement and a significant storage reduction. Open problems to be explored are to address data swamps and enable faster execution of advanced OLAP queries by developing: An advanced and intelligent data ingestion pipeline with knowledge extraction and linking capabilities A data profiling method to capture the significant and anomalous data to reduce storage footprint while enhancing data linking capabilities A knowledge graph to augment the metadata management in lakehouses

### *M. An Overview of Current Data Lake Architecture Models*

The paper aims to address the evolution of Data Lake architecture and the identification of their advantages. It highlights the various emerging architectures and their benefits in terms of data storage, analysis, and end-user consumption. The paper also covers the importance of employing Data Governance processes in the Data Lakes. The analyzed data lake models, the zone architecture, the layered architecture, and the Data Lakehouse architecture, have similarities and differences. The zone architecture focuses on data structures and pipelines between zones, while the layered architecture emphasizes the roles of different layers regardless of data structure. Both have transient zones for data reception and initial quality control, while the zone architecture explicitly defines raw data storage and the necessity of a metadata repository. The harmonized and distilled zones and the transformation layer have similarities to data marts, but the zone architecture has explicit data modeling. The interaction layer is like the exploration zone and provides users access to metadata. The zone architecture differs from the Data Lakehouse by holding all data in the same environment and not requiring a data virtualization layer. In the context of Data Lakes, data governance is crucial for effective data management. Without proper governance, a Data Lake can become a repository of noisy data, rendering it useless. As with Data Warehouses, Data Governance is essential for Data Lakes to ensure accurate and timely availability of data, particularly in complex architectural models where data lineage and metadata play a significant role. Future work is to focus on combining models to create a new architecture model and evaluating the impact of deployment options (on-premises, cloud, and hybrid) on the architecture choice.

### *N. Metadata Management on Data Processing in Data Lakes*

Data preparation is the most time-consuming part when analyzing data and modern metadata management in data lakes is not accommodative of process metadata and their reusability. To make data preparation more efficient and effective, this paper focuses on adding four aspects of data processing metadata on top of a generic metadata model that has been proposed in another paper: Process characteristics: the source and target dataset, name, creation date, and the user Process definition: processes are defined by a description and a set of keywords Technical information: the source code and execution details Process content: a set of coarse grained operations Operations are defined as DSoutput ← OP((DSinput1, [DSinput2...DSinputN ]), ARG) where: OP is the data transforming operation DSinput1, [DSinput2]...[DSinputN] is the source dataset(s) of the operation ARG is the argument of the operation, it can be a condition or a function of an operation DSoutput is the result of the operation To make operations compatible with classical ETL operations, the list of operations (OP) is: Filtering (FL) Formatting (FMT) Aggregation (AGGR) Calculation (CALC) Consolidation (CNSLD) Merging (MERGE) Join (JOIN) The data used to test the proposed model are from two projects that use the data lake of the University of Teaching Hospital (CHU) of Toulouse. The EHDEN Project which uses the electronic health record (EHR) as a data source and creates the OMOP dataset for collaborative analysis and the EBERS Project which analyzes all textual medical reports stored in the CHU database. One use case of the metadata was of a head trauma doctor looking to analyze all medical history of his patients which needs to be extracted from the EHR. Since the EHR database is not well documented, the metadata system was used to check if the reports have already been extracted elsewhere. The results showed that the hospitalization reports have been previously extracted for the EBERS project. This showed the effectiveness of the proposed metadata system. Two open problems are suggested. One is the automatic extraction of metadata and two is implementing a graphical and ergonomic interface for the metadata management system.

### *O. Implementing big data lake for heterogeneous data sources*

The paper is trying to address the challenges of collecting, storing, integrating, and analyzing heterogeneous data sources from different cities in a smart city platform, specifically in the CUTLER project, which aims to generate evidence-based solutions for policy decision-making related to coastal urban development using big data solutions. The paper proposes an approach to manage and acquire the data and presents the first version of the platform for data acquisition, storage, and analysis. The article proposes a data lake architecture that relies on Hadoop Ecosystem for data collection, storage, processing, analysis, and visualization. The proposed platform uses different big data technology tools and follows a data pipeline approach, including custom data collection, data ingestion, data storage, data exploration and analysis, and data visualization. The paper's limitations include insufficient support for data quality control and limited security measures such as restricted access to whitelisted IPs, requiring further research for more advanced security protocols and the involvement of cyber-security specialists for the final deployment of the platform. Future work includes experimenting with alternative data management and processing architectures, such as lambda-architecture, testing alternative frameworks for data ingestion, such as Apache Kafka or Apache NiFi, and integrating more advanced visualization tools like Kibana and Grafana for a richer user experience. Another future work mentioned is exploring better approaches for data harmonization and interoperability, such as enabling verification upon data models and experimenting with Semantic Web technologies, as well as exploring alternative metadata management instruments such as Apache Atlas or Kite SDK, as the current mechanisms place metadata management responsibilities on data lake developers.

### *P. A Zone Reference Model for Enterprise-Grade Data Lake Management*

The paper discusses the use of data lakes for analytics and proposes the use of different zones for storing data at different processing degrees to increase efficiency and reuse of processed data. The authors assess existing zone models and develop a reference model for enterprise-grade data lake management that meets the requirements derived from real-world industry use cases. The reference model specifies six zones and their characteristics and is evaluated through a prototypical implementation for a product lifecycle management analytics use case that covers multiple business domains and data sources. The assessment shows that the reference model is generally applicable and suitable for enterprise use. The six zones mentioned are: The Landing Zone is the first zone of the data lake where data is ingested as batch or stream from sources and then forwarded to the Raw Zone for permanent storage. Data remains mostly raw, with basic transformations allowed upon ingestion, and is governed, non-historized, and non-persistent, with no modeling approach defined. It is not intended for end-user access and may be implemented using Kafka to forward data to diverse storage systems and stream processing engines. The Raw Zone stores data persistently in mostly raw format with basic transformations and historization of changes, allowing data scientists to access and copy data for analytics. In the End Customer Service use case, all data is permanently stored in the Raw Zone, with various storage systems used to store data where it fits best and anonymization of personal data. The Harmonized Zone contains a copy of or a view on a subset of the data stored in the Raw Zone and provides a consolidated and standardized view of the data using a standardized modeling approach such as dimensional modeling or Data Vault. Data schema and syntax change compared to the source data, and master data management is of high importance. The Distilled Zone focuses on increasing efficiency by preparing data for specific use cases through complex processing, potentially changing the schema and granularity. It is use case-dependent and accessible to both data analytics experts and domain experts through easy-to-use interfaces, and in the End Customer Service analytics use case, it is modeled using Data Vault with added fields and pre-defined KPIs. The Explorative Zone is an unstructured and flexible area where data scientists can freely manipulate data to conduct any analyses they desire, with few restrictions or rules, but with sensitive data being handled with strict protocols, and results of analyses may be forwarded to the Distilled Zone. The Delivery Zone provides tailored data subsets for specific usage and applications, similar to data marts and operational data stores in data warehousing, and supports a large number and variety of users with little knowledge on data analytics. Future work is to focus on investigating and identifying implementation patterns and challenges for zones in a data lake, taking into account data modeling, storage architecture, and other aspects, with the aim of providing guidance for the definition of a standardized data lake architecture that enables interoperability with other systems.

### *Q. Data Lake Management: Challenges and Opportunities*

The paper is addressing the challenges and opportunities associated with the management of data lakes. The paper goes on to explain challenges and opportunities of data ingestion, data extraction, data cleaning, dataset discovery, metadata management, data integration, and dataset versioning. Modern data lakes support data ingestion from various sources, with bookkeeping and indexing being the main tasks. Shallow data sketches can be applied to maintain a basic organization of ingested datasets, but challenges remain in real-time ingestion of high-velocity data with more sophisticated indexing. Data ingestion creates raw datasets in a specific format while data extraction transforms this raw data into a predetermined data model. Extraction is well studied, yet opportunities remain for advancement and taking full advantage of the "wisdom of the lake" for future extractions. Cleaning data within the context of a data lake presents challenges due to schema-less heterogeneous formats and schemas that may only be specified at the application level. The CLAMS approach addresses this problem by enforcing quality constraints on raw heterogeneous lake data right after ingestion and extraction. The size of data and the absence of a comprehensive schema or data catalog has made data discovery a significant challenge in data lakes. Various techniques have been developed, including search-based and navigation-based discovery, as well as analysis-driven discovery. The absence of complete data catalogs in data lakes makes metadata management systems essential for on-demand discovery and integration, as well as raw data cleaning. Examples of metadata management systems include Google Dataset Search (GOODS), Constance, Ground, and Skluma, each with their unique features and approaches to metadata extraction and organization. The challenge remains to better extract and connect knowledge from lakes and incorporate it into existing knowledge bases. Traditional integration techniques are not useful for data lakes. On-demand integration is challenging as it requires finding relevant datasets and integrating them meaningfully, with techniques such as Infogather and schema mapping being potential solutions. Data lakes require effective versioning to handle the dynamic nature of data, but traditional distributed file systems are not adequate for this task. Efficiently storing and retrieving versions and managing versioning information are important features of a successful data lake system, and early approaches such as DataHub provide a git-like interface, but challenges remain in managing schema evolution, data formats, and version detection.

### *R. Data Lakes: Trends and Perspectives*

The paper is addressing the challenges of handling heterogeneous and voluminous data for Decision Support Systems (DSS) in the big data era, and the deficiencies of Data Warehouse (DW) solutions. The authors propose the concept of a data lake (DL) as a solution and aim to provide a more complete vision and generic architecture for DL, as well as identify major issues and future research directions, particularly in the context of health-care IT activities. The

paper proposes a functional architecture for data lakes containing four zones: Raw data zone, Process zone, Access zone, and Governance zone. The Raw data zone stores all types of data in their native format, the Process zone transforms data according to requirements, the Access zone provides data access for analytics, and the Governance zone ensures data security, quality, lifecycle, access, and metadata management. The article discusses the integration of a data lake (DL) in an enterprise's Information System (IS) alongside the commonly used Data Warehouse (DW). The authors propose that DLs should coexist with DWs because they have different objectives and users, and a DL cannot simply replace a DW. To propose such an ecosystem, research problems related to functional and technical architecture definitions and data refreshing and updating must be solved. The article then examines the importance of metadata management systems for data lakes, which store and process raw data. The proposed metadata classification includes categories such as dataset containment, provenance, content similarity, and security metadata, to support the verification of data lineage, access, and sensitivity. However, to propose an appropriate metadata management solution, research problems related to the conceptual schema, attributes, and storage of metadata, as well as the extraction of metadata from structured, semi-structured, and unstructured data, must be addressed.
The article also mentions that to ensure efficient and secure usage of data lakes, data governance is required, which involves policies, standards, and practices to manage data and associated processes. This governance can be classified into data assets and IT assets. Future research should focus on metadata and data quality management, data life-cycle management, and security/privacy in data lakes.

## III. Model description

### *A. Overview*

Our chosen methodology for constructing a data pipeline involves the use of Apache NiFi and Apache Kafka for data ingestion, and lakeFS for data management. The aim of this data pipeline is to proficiently process and transfer data from its origin to an output destination, such as a data lake or MongoDB database. Apache NiFi will orchestrate the data flow, whereas Apache Kafka will operate as the messaging platform handling data ingestion and delivery. To validate our pipeline, we will send data through the pipeline, message brokers, and output destinations, ensuring its functionality. To evaluate the pipeline's performance, we will gather metrics, including the duration of data ingestion, message throughput, and resource utilization. These experiments are intended to highlight potential bottlenecks, thus enabling us to optimize the pipeline for enhanced performance. The data management segment will employ lakeFS to manage and manipulate data. LakeFS provides a Git-like version control system for objects within a data lake. Our validation process will involve applying queries and version control to our data. This exercise will identify any functionalities that are either weak or absent within our data management system. By highlighting and addressing these issues, we aim to strengthen the overall robustness of our data pipeline.

### *B. Data*

We will be using the MobiAct dataset, which is a publicly available dataset of accelerometer and gyroscope sensor readings collected from mobile devices during various physical activities. The dataset contains sensor data for activities such as walking, jogging, sitting, standing, and ascending and descending stairs, among others. To collect the data, we will download the MobiAct dataset from the source website and extract the relevant sensor data.

### *C. Implementation*

Our implementation is split into two parts: the ingestion data pipeline and data management. The data pipeline uses Apache NiFi and Apache Kafka to process and transfer data from a source to an output destination. The pipeline involves using Apache NiFi to orchestrate the data flow, and Apache Kafka to handle data ingestion and delivery. The data pipeline consists of multiple steps, starting with the data source, which in our case is the MobiAct dataset. We use Apache NiFi to ingest the preprocessed data and transfer it to Apache Kafka, which serves as the messaging platform for our data pipeline. Apache Kafka handles the message queueing and delivery of data to the output destination, which in our case is a MongoDB database or a data lake. Nifi is a drag and drop system so all the functions only need to be connected with the right configuration in order to work. The data management system consists of samples from the MobiAct dataset and lakeFS. lakeFS provides Git-like version control on objects to allow for better management of data. Python is used to perform operations in lakeFS from creating repos and branches to managing version history of our data. A repo is created to house the test data which are CSV files that have been extracted from the MobiAct dataset. To test basic data analysis functions, branches are created to act as isolated environments where queries can be performed on the test data.

### *D. Implementation Environment*

The implementation and test environment for our data pipeline is a MacBook Pro with the following configuration:

- CPU: 2.3 GHz 8-Core Intel Core i9
- Memory: 16 GB 2667 MHz DDR4
- Graphics: AMD Radeon Pro 5500M 4 GB, Intel UHD Graphics 630 1536 MB

We use the following software and tools to build and test our data pipeline:

- Apache Kafka: Messaging platform for handling data ingestion and delivery
- Apache NiFi: Orchestration tool for managing the data flow.
- MongoDB: Database for storing and querying data

We chose to use Apache Kafka and Apache NiFi for their scalability and flexibility in handling large volumes of data. MongoDB was selected as the output destination for its ability to handle unstructured data and its ease of use.

The implementation and test environment for data management is a MacBook Pro with the following configurations:

- CPU: 3.1 GHz Quad-Core Intel Core i7

- Memory: 16 GB 2133 MHz LPDDR3
- Graphics: AMD Radeon Pro 560 4 GB, Intel HD Graphics 630 1536 MB

We use the following software for our data management system:

- LakeFS: Data version control

### *E. Validation*

A. Experiment 1: We will conduct an experiment to evaluate the performance and efficiency of our data pipeline through processing the MobiAct dataset. We will collect metrics such as message throughput, latency, and resource utilization to identify the performance. This can help with validating the pipeline can function with the data.

B. Experiment 2: To test the functionality of our data pipeline, we will conduct a series of tests to ensure that data is properly ingested, processed, and delivered to the output destination. Specifically, we will publish data to Kafka and consume it to validate that it has been properly processed and stored in MongoDB or local file. To do this, we will intentionally introduce a failure line after the Kafka message has been received, which will allow us to validate that data is correctly written to a local file. If we can confirm that data is being properly processed and stored in MongoDB or local file, then we can be confident that our data pipeline is functioning correctly and meets our objectives.

C. Experiment 3: To test the functionality of our data management, multiple data samples from the MobiAct dataset will be uploaded into a LakeFS repo. From there a test branch is created where we can apply queries to the objects then the object versions can be compared between the main branch and test branch by searching commit flags. Then we can merge the changes and check for version history by checking merge history.

## IV. Result and Discussion

### *A. Results*

The following figure shows the result of the pipeline in Apache Nifi.

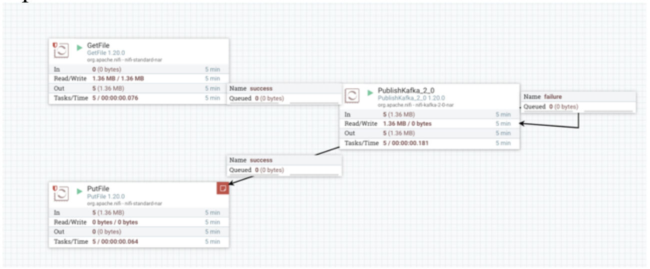

GetFile
GetFile 1.20.0
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 1.36 MB / 1.36 MB 5 min
Out 5 (1.36 MB) 5 min
Tasks/Time 5 / 00:00:00.076 5 min
Name success
Queued 0 (0 bytes)
PublishKafka_2_0
PublishKafka_2_0 1.20.0
org.apache.nifi - nifi-kafka-2-0-nar
In 5 (1.36 MB) 5 min
Read/Write 1.36 MB / 0 bytes 5 min
Out 5 (1.36 MB) 5 min
Tasks/Time 5 / 00:00:00.181 5 min
Name failure
Queued 0 (0 bytes)
Name success
Queued 0 (0 bytes)
PutFile
PutFile 1.20.0
org.apache.nifi - nifi-standard-nar
In 5 (1.36 MB) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 5 / 00:00:00.064 5 min

**Fig. 1.** Publish Kafka

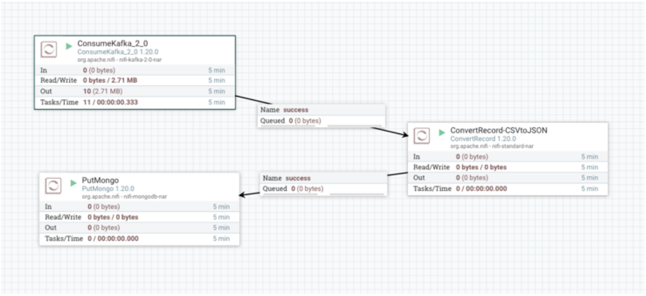
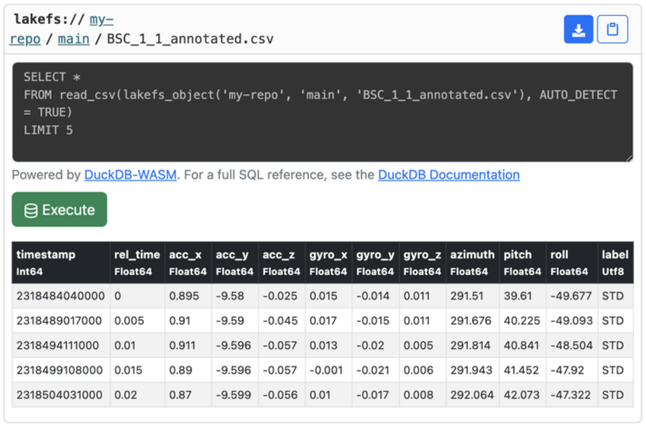

ConsumeKafka_2_0
ConsumeKafka_2_0 1.20.0
org.apache.nifi - nifi-kafka-2-0-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 2.71 MB 5 min
Out 10 (2.71 MB) 5 min
Tasks/Time 11 / 00:00:00.333 5 min
Name success
Queued 0 (0 bytes)
ConvertRecord-CSVtoJSON
ConvertRecord 1.20.0
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Name success
Queued 0 (0 bytes)
PutMongo
PutMongo 1.20.0
org.apache.nifi - nifi-mongodb-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min

**Fig. 2.** Consumer Kafka

The pipeline can read the data with a speed of Min 277.90 KB, Max 1.36 MB, Mean 1.26 MB, write the data with Min 0.00 bytes, Max 74.63 MB, Mean 2.71 MB. The result of experiment 3 was a successful creation of a test branch and simple queries were applied to objects, however, that was the extent of the results. Simple queries were able to be performed, but new objects were not able to be created from the manipulation of existing objects.


lakefs:// my-repo / main / BSC_1_1_annotated.csv

```
SELECT *
FROM read_csv(lakefs_object('my-repo', 'main', 'BSC_1_1_annotated.csv'), AUTO_DETECT
= TRUE)
LIMIT 5
```

Powered by DuckDB-WASM. For a full SQL reference, see the DuckDB Documentation

| timestamp Int64 | rel_time Float64 | acc_x Float64 | acc_y Float64 | acc_z Float64 | gyro_x Float64 | gyro_y Float64 | gyro_z Float64 | azimuth Float64 | pitch Float64 | roll Float64 | label Utf8 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2318484040000 | 0 | 0.895 | -9.58 | -0.025 | 0.015 | -0.014 | 0.011 | 291.51 | 39.61 | -49.677 | STD |
| 2318489017000 | 0.005 | 0.91 | -9.59 | -0.045 | 0.017 | -0.015 | 0.011 | 291.676 | 40.225 | -49.093 | STD |
| 2318494111000 | 0.01 | 0.911 | -9.596 | -0.057 | 0.013 | -0.02 | 0.005 | 291.814 | 40.841 | -48.504 | STD |
| 2318499108000 | 0.015 | 0.89 | -9.596 | -0.057 | -0.001 | -0.021 | 0.006 | 291.943 | 41.452 | -47.92 | STD |
| 2318504031000 | 0.02 | 0.87 | -9.599 | -0.056 | 0.01 | -0.017 | 0.008 | 292.064 | 42.073 | -47.322 | STD |

**Fig. 3.** Simple query on an object in lakeFS web-UI

### *B. Discussions*

The error we experienced while transferring data from Kafka to MongoDB is most likely attributable to a mismatch between the schema of the CSV data and the anticipated JSON schema for MongoDB. The Apache NiFi processor, which is responsible for converting the CSV data to JSON, might have encountered difficulties locating the expected schema. In order to resolve this issue, we should consider exploring avenues for managing schema changes within the data pipeline.

One potential remedy could involve utilizing a schema registry to manage and version schemas for the data. This could significantly aid in tracking data changes over time.

The inability to manipulate objects may stem from the limited capabilities of the integrated DuckDB in lakeFS, which likely lacks the functionality to perform advanced queries. To circumvent this challenge, we could manipulate objects managed in lakeFS using external software. This software would have the capacity to query objects in lakeFS through connections established using Python scripts, thereby providing the ability to perform advanced manipulations.

## V. Conclusion And Future Work

### *A. Limitation*

Despite our success in implementing the data pipeline, there were distinct limitations encountered during our work. Although we carried out experiments to assess the performance and efficiency of our data pipeline, a more comprehensive testing and benchmarking process involving larger datasets and a range of configurations is required for complete validation.

One particular challenge encountered was the suboptimal connection between the pipeline and MongoDB, which necessitates further refinement. Additionally, the capabilities of our data management system posed a constraint, particularly in regards to data manipulation. Therefore, future work should address these limitations, aiming to improve the integration between pipeline and database, and enhancing the data manipulation capacity of the data management system.

### *B. Future Work*

There are several promising directions for future work in the domain of data pipelines. Firstly, we could consider linking the data pipeline with the data lake we are constructing. This integration would enhance the overall flow and accessibility of data across the system. Secondly, the issue of handling schema changes during ingestion still requires attention. It's crucial to ensure that our pipeline remains adaptable to fluctuations in data structures, maintaining its performance irrespective of such changes. As for data management, we could explore the use of an external system for querying, such as Apache Spark. We could employ data frames written with PySpark to ensure a seamless interaction with our data. This approach would harmonize with our current implementation, where we use Python for versioning data with LakeFS. By focusing on these areas, we can continue to enhance the flexibility, efficiency, and overall performance of our data pipeline, further advancing our capabilities in data processing and analysis.

## Acknowledgment

The preferred spelling of the word "acknowledgment" in American English is without an "e" after the "g." Use the singular heading even if you have many acknowledgments. Avoid expressions such as "One of us (S.B.A.) would like to thank ... ." Instead, write "F. A. Author thanks ... ." In most cases, sponsor and financial support acknowledgments are placed in the unnumbered footnote on the first page, not here.

## References

[1] Ahmet, A., & Abdullah, T. (2020). Real-time social media analytics with deep transformer language models: a big data approach. In 2020 IEEE 14th International Conference on Big Data Science and Engineering (BigDataSE) (pp. 41-48). IEEE.

[2] Giebler, C., Gröger, C., Hoos, E., Schwarz, H., & Mitschang, B. (2020, October). A zone reference model for enterprise-grade data lake management. In 2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC) (pp. 57-66). IEEE.

[3] Harby, A. A., & Zulkernine, F. (2022, December). From Data Warehouse to Lakehouse: A Comparative Review. In 2022 IEEE International Conference on Big Data (Big Data) (pp. 389-395). IEEE.

[4] Hiraman, B. R., Viresh, C., & Abhijeet, K. C. (2018). A Study of Apache Kafka in Big Data Stream Processing. 2018 International Conference on Information, Communication, Engineering and Technology (ICICET), (pp. 1-3). IEEE.

[5] Hlupić, T., Oreščanin, D., Ružak, D., & Baranović, M. (2022, May). An Overview of Current Data Lake Architecture Models. In 2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO) (pp. 1082-1087). IEEE.

[6] Liu, R., Isah, H., & Zulkernine, F. (2020, September). A big data lake for multilevel streaming analytics. In 2020 1st International Conference on Big Data Analytics and Practices (IBDAP) (pp. 1-6). IEEE.

[7] Lv, H., Zhang, T., Zhao, Z., Xu, J., & He, T. (2020). The Development of Real-time Large Data Processing Platform Based On Reactive Micro-Service Architecture. 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), (pp. 2003-2006). IEEE.

[8] Megdiche, I., Ravat, F., & Zhao, Y. (2021). Metadata management on data processing in data lakes. In SOFSEM 2021: Theory and Practice of Computer Science: 47th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2021, Bolzano-Bozen, Italy, January 25–29, 2021, Proceedings 47 (pp. 553-562). Springer International Publishing.

[9] Mehmood, H., Gilman, E., Cortes, M., Kostakos, P., Byrne, A., Valta, K., ... & Riekki, J. (2019, April). Implementing big data lake for heterogeneous data sources. In 2019 ieee 35th international conference on data engineering workshops (icdew) (pp. 37-44). IEEE.

[10] Nargesian, F., Zhu, E., Miller, R. J., Pu, K. Q., & Arocena, P. C. (2019). Data lake management: challenges and opportunities. Proceedings of the VLDB Endowment, 12(12), 1986-1989.

[11] Pal, G., Li, G., & Atkinson, K. (2018). Big data real time ingestion and machine learning. In 2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP) (pp. 25-31). IEEE.

[12] Ravat, F., & Zhao, Y. (2019). Data lakes: Trends and perspectives. In Database and Expert Systems Applications: 30th International Conference, DEXA 2019, Linz, Austria, August 26–29, 2019, Proceedings, Part I 30 (pp. 304-313). Springer International Publishing.

[13] Sharma, G., Tripathi, V., & Srivastava, A. (2021). Recent Trends in Big Data Ingestion Tools: A Study. In Research in Intelligent and Computing in Engineering (pp. 873-881). Springer, Singapore.

[14] Shree, R., Choudhury, T., Gupta, S. C., & Kumar, P. (2017). KAFKA: The modern platform for data management and analysis in big data domain. 2017 2nd International Conference on Telecommunication and Networks (TEL-NET), (pp. 1-5). IEEE.

[15] Surekha, D., Swamy, G., & Venkatramaphanikumar, S. (2016). Real time streaming data storage and processing using storm and analytics with Hive. 2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT), (pp. 606-610). IEEE.

[16] Ta, V. D., Liu, C. M., & Nkabinde, G. W. (2016). Big data stream computing in healthcare real-time analytics. 2016 IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), (pp. 37-42). IEEE.

[17] Vyas, S., Tyagi, R. K., Jain, C., & Sahu, S. (2022). Performance Evaluation of Apache Kafka – A Modern Platform for Real Time Data Streaming. 2022 2nd International Conference on Innovative Practices in Technology and Management (ICIPTM), (pp. 465-470). IEEE.

[18] Wu, H., Shang, Z., & Wolter, K. (2020). Learning to Reliably Deliver Streaming Data with Apache Kafka. 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), (pp. 564-571). IEEE.