Penn State University

Great Valley Campus

Engineering Division

Data Specification for

*Real Time Distributed System for Trend Analysis*

Version 1.0

03/23/2025 – 04/30/2025

**Table of Contents**

[Introduction 3](#_Toc197115293)

[Purpose 3](#_Toc197115294)

[Project Summary 3](#_Toc197115295)

[Requirements Definition 4](#_Toc197115296)

[Considerations 6](#_Toc197115297)

[Document Change Log 6](#_Toc197115298)

[2. Architecture Design 7](#_Toc197115299)

[2.1 Overall System Architecture 7](#_Toc197115300)

[2.2 Long Term Storage Database 7](#_Toc197115301)

[2.3 Real Time Storage Database 8](#_Toc197115302)

[2.4 Streaming System 9](#_Toc197115303)

[2.5 Reflective analysis of the real time system architecture. 10](#_Toc197115304)

[3. Data Processing 11](#_Toc197115305)

[3.1 Long Term Storage Database 11](#_Toc197115306)

[3.2 Real Time Storage Database 13](#_Toc197115307)

[3.3 Streaming System 17](#_Toc197115308)

[3.4 Reflective analysis data processing in real time system. 18](#_Toc197115309)

[4. Reporting System 19](#_Toc197115310)

[4.3 Reflective analysis of reporting for real time systems. 19](#_Toc197115311)

[Conclusions 21](#_Toc197115312)

Real Time Distributed System for Trend Analysis  
Rahul Veerapur

# Introduction

This project focuses on the design and implementation of real time, distributed system that analyzes news trends related to universities in Pennsylvania. By leveraging tools such as Kafka, Cassandra, MongoDB. The System processing and stores live data streams from the Googe News Api via RapidApi, delivering timely insights into media coverage.

The system aims to address keys questions such as: What are the trending topics surrounding Pennsylvania universities ? and which new organizations are most active in reporting on these institutions? This real time pipeline ensures that the systems can scale with incoming data, respond to new trends instantly and support both exploratory analysis and long-term monitoring.

Kafka handles the continuous ingestion of news article while mongodb and Cassandra serve as robust storage solution for unstructured and time series data respectively for which the system enables us to extract data , process it and give us an modular analysis.

This project demonstrates the application of distributed computing and system design principles which makes it possible to scale the framework.

# Purpose

The purpose of this project is to implement a real time system that monitors collects and analyzes news articles in real time offering trends to emerging markets across the media coverage. The systems also aims to provide immediate access to trending news topics by organizing into the groups.

# Project Summary

This paragraph is used to introduce the following subsections, which can be used for an executive level overview.

1. **Objectives**

* To develop a real time distributed system.
* To analyze and extract trending topics from news articles.
* To identify the most frequently contributing news organization.
* To design a scalable and fault tolerant architecture.
* To ensure low latency data processing.
* Validation of the architecture.
* To visualize the reports findings.

1. **Scope**

* Streaming layer which uses Kafka to scale the communication between apis and the processer.
* Processing layer where docker is used as an microservice layer responsible for parsing and enriching the data such as source activity and other determined variables.
* Storage layer here 2 will be implemented such as Cassandra and mongodb , where the first once is for handling high throughput time series storage of structured metrices and mongodb is used for storing unstructured high throughput directly from the processing layer.
* System characteristics emphasis is in distributed system qualities including scalability, reliability , resilience and horizontal scaling.
* Exclusion such as UI and analytics for the end results

1. **References**

* A Scalable and robust framework for data stream ingestion link: <https://arxiv.org/abs/1812.04197>
* Real time streaming system for evolving graphs to support sub-millisecond per-update analysis at millions op : <https://arxiv.org/abs/2004.00803>

1. **Outstanding Issues**

* Underutilization of stream processing frameworks where many research papers focus more about batch processing reducing responsiveness and scalability.
* Limited integration of distributed storage system.
* Scalability and fault tolerance issues.
* Lack of practical toolchain validation.

# Requirements Definition

1. **Goals**

* Develop a robust real time data ingestion system using Apache Kafka to collect news articles continuously from apis.
* Implement distributed processing services using python.
* Use Cassandra for time series structured data such as keyword counts and timestamps.
* Use mongodb for time series unstructured document to store the entire articles.
* Ensure horizontal scalability and high availability of all the system components via containerization and stateless design.
* Implement a modular architecture that can be extended to include additional data source, analytics modules and dashboards.

1. **Usability Requirements**

* The system shall require minimal setup beyond initial credentials and topic filters.
* Data will be processed and available within seconds of ingestion.
* New sources processor and database should be easy to add through modular code structure.
* The system must provide clear logs for system and message flows to ensure traceability for each modules.

1. **System Security Requirements**

* Kafka communication should be encrypted and authenticated using protocols for the pipelines.
* Role base access control should be provided individually for Cassandra and mongodb separately.
* Docker images should be minimal and scanned regularly for vulnerabilities If deployed via cloud.

1. **Business Questions**

* What are the trends in news topics related to universities in Pennsylvania ?
* What are the most frequently posted news organizations that post articles on universities in Pennsylvania?

1. **Data Requirements**

* Data sources such as news articles either via api or web sources.

1. **Design Constraints**

* Hardware limitation where the given number of nodes is not suitable to achieve near 0 real time analytics
* Data volume control where the topic filter must be tight to prevent overwhelming the system with unnecessary data.
* Cassandra has limitation in adhoc querying.
* Performance affected by the external apis.

# Considerations

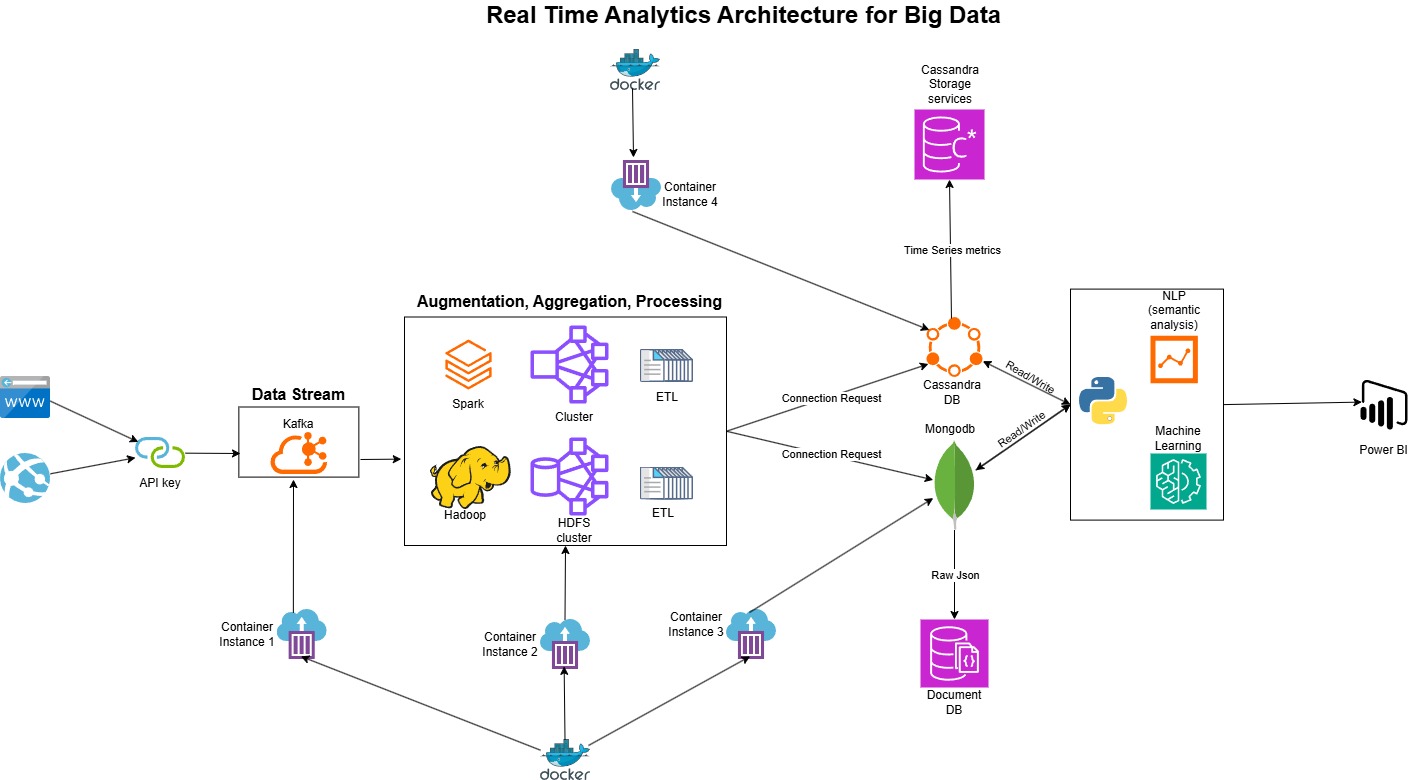
(May include as separate document)

# Document Change Log

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Change Date** | **Version** | **CR #** | **Change Description** | **Author and Organization** |
| 03/29/2025 | 1.0 |  | Project Creation | Rahul Veerapur |
| 04/24/2025 | 1.2 |  | Draft Creation | Rahul Veerapur |
| 05/02/2025 | 2.0 |  | Result additions | Rahul Veerapur |
| 05/03/2025 | 2.0.1 |  | Added data insertion via api (added changes to initial draft) | Rahul Veerapur |
| 05/03/2025 | 2.1 |  | Final formatting | Rahul Veerapur |

# 2. Architecture Design

## 2.1 Overall System Architecture



Api structure : The news articles used in this project are retrieve from the google news api via the rapidapi platform this aggregates news content from various online news sources and provides a convenient way to query current articles based on specific keywords. In this project the keywords search are Pennsylvania universities or Penn state

This architecture represents a real-time, distributed analytics system designed for processing large-scale news data. Data is collected from web APIs using API keys and streamed into Kafka, which acts as a high-throughput message broker. Docker containers manage isolated deployments of Kafka, Spark, Hadoop clusters, and storage systems to ensure scalability and reliability. Spark Streaming applications running on a distributed cluster perform real-time ETL (Extract, Transform, Load) operations, cleaning and preparing the data. Processed data is stored in MongoDB for raw JSON documents and Cassandra for time-series analytics metrics. Further, machine learning and NLP models analyze the data for trends, sentiment, and entity extraction. The final insights are visualized through tools like Power BI, enabling real-time dashboards. This architecture combines real-time stream processing, distributed computing, machine learning, and scalable storage, all orchestrated through containerization for flexible deployment.

## 2.2 Long Term Storage Database

In the system we use **MongoDB as the long-term storage solution** to retain all the raw, unprocessed news data that comes in from the APIs. Since MongoDB is a document-oriented NoSQL database, it works really well for this use case because it allows us to store each news article in its original JSON format without needing to define a rigid schema. News articles often come with different structures some might include authors, tags, locations, or images, while others might not. MongoDB gives the flexibility to store all of that variation naturally, which makes it a perfect fit for archiving raw data.

One of the main reasons we chose MongoDB for long-term storage is because it allows us to **preserve the original form of the data**, even after it has gone through real-time processing. That’s important because sometimes there might need to go back and reprocess the data using a better NLP or machine learning model in the future. By keeping the raw data safe in MongoDB, we can always rerun analytics or generate new insights without depending on external APIs again. It also helps from a traceability point of view if something unusual shows up in my analytics, there can trace it back to the original article and verify what actually happened.

Another big advantage is how **powerful and flexible the querying is in MongoDB**. We can write queries that search articles by keywords, filter by date ranges, or even use full-text search to find topics. This makes it easy for us to dig into the data and generate insights manually or as part of a scheduled analytics pipeline. I also find it useful when running batch jobs that analyze historical data for trends across months or years.

As scalability was also a factor. MongoDB can handle large amounts of data through sharding, which means we can perform distribution of the data across multiple nodes as it grows. Since the whole system is containerized with Docker, we can scale MongoDB up easily by running more container instances as needed. It also integrates well with Spark, so I can feed historical data from MongoDB into my machine learning workflows when needed.

## 2.3 Real Time Storage Database

In the system we can use **Cassandra as the real-time storage database** to store the structured, processed data that comes out of the Spark streaming pipeline. Cassandra is a distributed, NoSQL database that’s optimized for high-speed writes and fast, scalable read operations. It's especially well-suited for storing **time-series data**, which is exactly what we need for capturing metrics like keyword trends, article counts over time, or sentiment scores for different universities.

Once the raw news articles are processed in real time using Spark where we extract keywords, perform sentiment analysis, and tag the articles the output is immediately written to Cassandra. We chose Cassandra because of its ability to **handle a high volume of concurrent writes with very low latency**, which is critical in a real-time system where new data is constantly flowing in. The data model we use in Cassandra is designed around time windows, so we can quickly query things like “top trending topics this hour” or “number of articles published about a specific university today.”

Cassandra is designed for distributed environments, so it scales horizontally. As the data grows, we can just add more nodes to the cluster without worrying about degrading performance. This is important for a long-running system like this system, which continuously collects and analyzes streaming data. Since this entire system is containerized with Docker, scaling Cassandra is also straightforward where we can add more instances and let the database handle the replication and data distribution.

Another important point is fault tolerance. Cassandra’s architecture is masterless and uses replication across nodes, so if one node goes down, others can continue serving the data. This adds reliability to the system, which is especially important for real-time analytics where uptime and consistency matter. Cassandra also integrates well with the rest of the system Spark can write to it directly, and it acts as a fast-access layer for dashboards and any downstream analytics tools.

## 2.4 Streaming System

In the implemented system Apache Kafka serves as the backbone of the entire streaming infrastructure. It’s responsible for managing the continuous inflow of news data from external sources like the Google News API and making sure that data is reliably passed to all the necessary downstream components for processing, storage, and analytics. Kafka enables the system to handle real-time events efficiently, while also ensuring fault tolerance, scalability, and low latency.

The system starts by fetching news articles from different APIs at regular intervals. Once fetched, these articles are published to specific Kafka topics for example, where might have a topic. From there, Kafka acts as **a** high-throughputmessage broker, making the data available to multiple consumers simultaneously. One consumer reads the data and pushes it into MongoDB for long-term archival, another sends it into a Spark cluster for real-time processing, and others could handle logging or alert generation. This publish-subscribe model is essential because it decouples data producers from consumers, making the system modular and easier to maintain or expand.

One of the key advantages of using Kafka is its persistence and durability. Kafka stores all published messages on disk and can retain them for a configurable time period. This is incredibly useful in a real-time system because if a component like Spark or Cassandra goes down or restarts, it won’t miss any data it just resumes consuming from where it left off. Kafka also provides message offsets, allowing precise control over how consumers process data and enabling replaying data for debugging, testing, or reprocessing.

Kafka is also horizontally scalable. Where we can increase throughput just by adding more brokers or partitions. Since the system is Dockerized, I run Kafka brokers in containers, and we can scale them up or down depending on the incoming data volume. Similarly, consumers are also containerized, allowing for elastic scaling of processing power. This flexibility is crucial in a semester-long system that might have bursts of activity or evolve to handle more news sources.

Another important feature is Kafka’s low-latency communication between components. It ensures that the delay between when a news article is fetched and when it gets processed or stored is kept to a minimum — which is essential for timely analytics and real-time dashboards. Kafka also integrates well with Apache Spark through Kafka-Spark connectors, which allows Spark jobs to continuously listen to new data and process it instantly without polling or delays.

## 2.5 Reflective analysis of the real time system architecture.

Using Apache Kafka as the message broker turned out to be a critical choice. The importance of a streaming backbone that could persist messages and decouple components. As the system grew, Kafka’s durability and scalability became central to keeping the architecture resilient and modular. Components such as offsets, partitions, and topic replication contribute not only to fault tolerance but also to performance when multiple consumers are working in parallel.

Another major takeaway was around real-time processing using Spark Structured Streaming. Spark made it easier to process incoming articles on the fly applying transformations, extracting entities, performing keyword analysis, and computing trends all in a streaming context. However, setting up Spark in a distributed environment, tuning batch intervals, and ensuring backpressure handling made streaming jobs can quickly become complex when the data volume increases. Balancing real-time responsiveness with resource consumption was a continuous challenge.

On the storage side, there was a clear difference in purpose between Cassandra and MongoDB. Cassandra’s write-optimized, high-speed capabilities made it ideal for storing analytics-ready, real-time metrics. MongoDB, on the other hand, allowed us to store raw, unstructured historical data with complete flexibility. It was interesting to see how both databases served different but complementary roles. This also gave deeper appreciation for polyglot persistence in distributed systems picking the right tool for the right job rather than forcing a one-size-fits-all database.

Containerizing the system using Docker made deployment and testing far more manageable. Which enabled us to simulate a distributed setup locally, debug issues in isolation, and scale components independently.

3. Data Processing

## 3.1 Long Term Storage Database

A computer screen shot of text

AI-generated content may be incorrect.

The figure above shows a sample document from the MongoDB collection. MongoDB is used as a long-term storage solution to retain raw data, with the schema directly derived from the fields provided by the API. Each document includes an id, which is automatically generated when added to the collection; a title, representing the news headline; a newsURL, which is the link to the article obtained through the API via the Kafka cluster; an image field, containing the URL of an associated image if available on the news website; and a publisher, indicating the source or organization responsible for the news article.

A screen shot of a computer program

AI-generated content may be incorrect.

This image shows the connection configuration used to connect to MongoDB. Since the script is executed within a Docker container, we substitute the IP address with localhost. The value 27017 represents the default port used to establish the connection with the MongoDB instance.

A screen shot of a computer program

AI-generated content may be incorrect.

Since Kafka receives data from the API and converts it into JSON format, pushing this JSON directly into MongoDB simplifies the data ingestion process. MongoDB's flexible schema design allows it to seamlessly handle changes in the data structure, such as the addition of new fields, without requiring predefined schemas or manual updates making it ideal for dynamic, real-time data storage.

A screen shot of a computer

AI-generated content may be incorrect.

The image above illustrates the real-time updating of fields in the system. However, since news related to Penn University is relatively infrequent, real-time changes may not be immediately visible. To achieve a truly real-time system, additional resources need to be allocated to Kafka, including scaling up its processing capabilities and increasing the number of clusters to handle higher data throughput more efficiently.

## 3.2 Real Time Storage Database

A screen shot of a computer

AI-generated content may be incorrect.

This image displays all the database names along with the number of clusters assigned to each, referred to as the replication factor. In this case, the database used is final\_stream, which contains the table realtime\_news. This can also be created from outside the Cassandra container via the connection to the container which contains python.

A screen shot of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

This image shows the schema for the realtime\_news table. The schema outlines the structure of the data being stored, including fields such as the news title, publication timestamp, source URL, publisher, and any associated media links. This schema is designed to support efficient querying and filtering of real-time news articles. It also ensures consistency in data organization while allowing flexibility to accommodate new fields if the incoming data structure evolves over time.

A screen shot of a computer

AI-generated content may be incorrect.

The above code snippet illustrates the process of inserting real-time news articles into the realtime\_news table within a Cassandra database. A prepared statement is defined to structure the insert operation, targeting fields such as newsurl, title, snippet, publisher, timestamp, and processed. The store\_in\_cassandra(article) function executes this statement by extracting the relevant data from each article dictionary. The timestamp is converted into a proper datetime format using datetime.fromisoformat() to ensure consistency, while the processed field is set to False to mark new, unprocessed entries. This method ensures high efficiency and reliability for data ingestion in a distributed Cassandra environment. Additionally, the function logs a brief preview of each processed article title and handles exceptions gracefully to support fault tolerance in the real-time data pipeline.

A screenshot of a computer

AI-generated content may be incorrect.

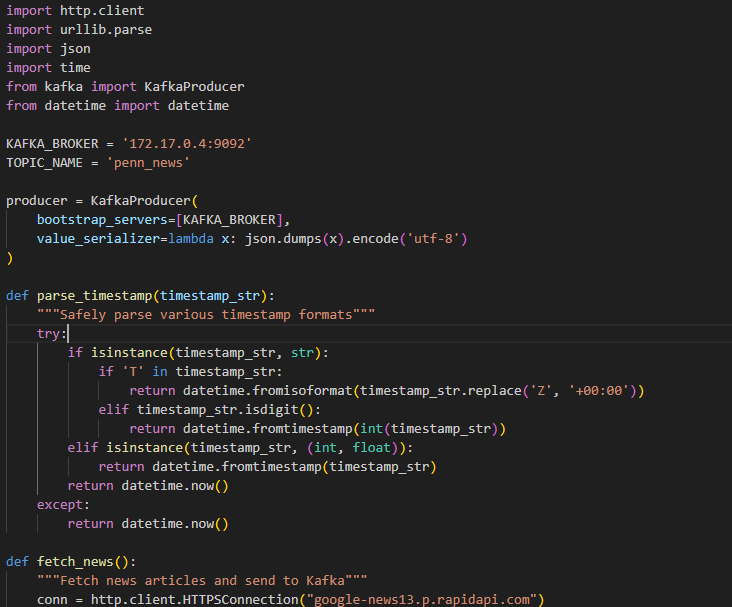
The image above shows the real-time storage database powered by Cassandra, which is continuously updated based on incoming Kafka topics. Cassandra is specifically designed for high write throughput, making it well-suited for real-time applications that require fast, continuous data ingestion. It follows a horizontally scalable architecture, meaning new nodes can be added to the cluster to handle increased load without any downtime. This distributed nature ensures that data is replicated across multiple nodes, offering high availability and fault tolerance. In our setup, as new topics are consumed from Kafka, the relevant data is written directly into Cassandra, enabling low-latency access and scalable real-time analytics.

## 3.3 Streaming System

A screenshot of a computer

AI-generated content may be incorrect.

The above image illustrates all the containers currently running using Docker. Each container plays a specific role in the architecture of the real-time data pipeline. The psu-cassandra-container runs the Cassandra database server, which serves as the real-time, horizontally scalable storage system for processed data. The psu-mongodb-container hosts the MongoDB server, used for storing raw data ingested directly from external sources. The Kafka container is responsible for managing the producer-consumer data stream. It collects news data via API calls, publishes it to defined topics, and delivers it to consumer services for further processing and storage. This containerized architecture ensures modularity, scalability, and ease of deployment across various environments.



The above code snippet illustrates the configuration and implementation of a Kafka producer in Python, which is responsible for sending real-time news data to a Kafka topic. The Kafka producer is initialized using the KafkaProducer class, with the broker address set to 172.17.0.4:9092, and the topic name defined as 'penn\_news'. The producer serializes the data into JSON format before sending it to Kafka.

A helper function named parse\_timestamp is defined to handle various timestamp formats robustly. Given the diversity of timestamp representations that may be received from external APIs, this function ensures that they are consistently parsed into Python datetime objects. It first checks if the input is a string. The function also supports direct integer and float inputs, converting them to datetime using fromtimestamp. If parsing fails at any point, the function defaults to returning the current datetime to ensure continuity in the data pipeline.

This preprocessing step is crucial for maintaining time consistency across records and supports downstream analytics and storage in Cassandra or MongoDB. The fetch\_news function is intended to pull news articles from the RapidAPI endpoint and push them into the Kafka stream for real-time processing.

A black screen with white text

AI-generated content may be incorrect.

This images shows the topics present in the Kafka.

## 3.4 Reflective analysis data processing in real time system.

In designing a real-time data processing pipeline using Kafka for high-throughput streaming, the system demonstrated seamless event flow and efficient message handling. Kafka's event-driven architecture enabled reliable data transmission, while Cassandra's write-optimized schema supported rapid data ingestion at scale. Simultaneously, MongoDB's horizontally scalable architecture effectively managed embedded JSON documents, allowing for flexible schema design and efficient raw data storage. Throughout the pipeline, data consistency was maintained across all three systems, ensuring synchronized and accurate updates. The overall system delivered strong cost-performance optimization under varying workloads, balancing speed, durability, and responsiveness. Additionally, it exhibited robust monitoring capabilities, effectively managing traffic spikes during peak loads. However, the full potential of this highly scalable architecture is realized only when deployed with sufficient computing and network resources to support its distributed nature.

4. Reporting System

## 4.3 Reflective analysis of reporting for real time systems.

A graph with different colored bars

AI-generated content may be incorrect.

This graphs shows the number of frequent publishers in the collected data.

A close up of words

AI-generated content may be incorrect.

This graph shows the world cloud of all the collected data here we can the terms Pennsylvania, education, college and universities as the search and collection was concentrated around this.

A blue circle with orange triangle and black text

AI-generated content may be incorrect.

Using the bag of words as a base we were able to clearly identify the news as if there were any incident or not and here we can see that only a small number of publication included news such as incident.

A blue and orange pie chart

AI-generated content may be incorrect.

Using the bag of words and NLTK we were able to classify those news which are educational to non educational.

Conclusions

This project successfully demonstrated the design and implementation of a real-time data processing pipeline integrating Kafka, MongoDB, and Cassandra to collect, store, and process university-related news articles. By using Kafka’s event-streaming capabilities, MongoDB’s flexible document model for raw data storage, and Cassandra’s high-throughput, write-optimized architecture for processed data, the system achieved a balanced solution for speed, scalability, and resilience. The pipeline maintained data consistency across components and handled varying loads efficiently, showcasing strong real-time performance and reliability. Despite the successful integration and functional outcomes, the system's performance is highly dependent on the availability of adequate computing resources and careful configuration of infrastructure components. Overall, this architecture presents a scalable and modular framework that can be expanded for broader real-time analytics applications across different domains.