UTM
UNIVERSITI TEKNOLOGI MALAYSIA

SECP3133 High Performance Data Processing

Semester 2 2024/2025

Project 2

Real-Time Sentiment Analysis of YouTube Comments Using Apache Architecture

SECTION	02
GROUP	1
GROUP MEMBERS	1. NEO ZHENG WENG (A22EC0093) 2. NURUL ERINA BINTI ZAINUDDIN (A22EC0254) 3. MUHAMMAD SAFWAN BIN MOHD AZMI (A22EC0221) 4. NUR ARINI FATIHAH BINTI MOHD SABIR (A22EC0244)
LECTURER	DR. ARYATI BINTI BAKRI
SUBMISSION DATE	11 JULY 2025

Table of Content

1.0 Introduction

This section introduces the project, which focuses on developing a real-time sentiment analysis pipeline in Apache architecture for YouTube comments related to Malaysian content. The project includes training a sentiment machine learning model, building a scalable pipeline with Apache Kafka and Spark, classifying sentiment, and visualizing trends over time in a real-time Kibana dashboard. The scope is limited to real-time analysis of YouTube comments on Malaysian topics.

1.1 Background

This project aims to design and implement a scalable real-time sentiment analysis pipeline for processing YouTube comments related to Malaysian content. The primary objective is to extract and analyze public sentiment in response to videos covering topics such as politics, history, culture, and social issues. By categorizing user comments into positive, negative, or neutral sentiments, the system offers a data-driven perspective on audience engagement and emotional response.

Real-time sentiment analysis holds significant value in the Malaysian context. It enables content creators, marketers, media analysts, and policymakers to monitor evolving public discourse, assess reactions to key events, and align content strategies accordingly. This timely feedback mechanism supports more responsive and informed decision-making.

1.2 Objectives

This project focuses on the development of a real-time sentiment analysis system designed to monitor and interpret public sentiment from Malaysian YouTube comments. The primary objectives are:

- To build an end-to-end data pipeline capable of ingesting, processing, and analyzing large volumes of YouTube comment data in real time using big data technologies such as Apache Kafka and Apache Spark.

- To accurately classify public sentiment into three categories by positive, negative, and neutral based on the textual content of user comments, using trained machine learning

models.

- To track and visualize sentiment trends associated with specific videos over time, enabling stakeholders to observe how public opinion evolves in response to different content themes, events, or viral topics.

1.3 Scopes

This project focuses on delivering a functional real-time sentiment analysis pipeline with specific boundaries to ensure feasibility and achieve the stated objectives. The scope encompasses the following key areas:

- Data Source: The project's data ingestion is exclusively limited to public YouTube comments. Data from other social media platforms (e.g., Twitter, Facebook), news articles, or other web sources are outside the current scope.
- Geographic and Content Focus: The sentiment analysis is specifically tailored to content related to Malaysian topics, as defined by the keywords used for video and comment extraction.
- Real-Time Processing: The system is designed for real-time ingestion and processing of newly generated YouTube comments. The scope does not include the backfilling or batch processing of extensive historical YouTube comment archives beyond what is captured by the continuous real-time stream.
- Sentiment Classification: The project focuses solely on three-category sentiment classification (positive, negative, neutral) based on textual content. More granular sentiment scores, emotion detection (e.g., anger, joy, sadness), or other advanced Natural Language Processing (NLP) tasks (e.g., entity recognition, topic modeling, summarization) are beyond this project's scope.
- Technology Stack: The implementation is confined to the specified big data technologies: Apache Kafka for message queuing and Apache Spark (specifically Spark Streaming) for real-time data processing and model inference.

2.0 Data Acquisition & Preprocessing

This section details the origin, volume, and characteristics of the dataset utilized for sentiment analysis in this project, along with the tools and methods employed for data cleaning and preprocessing.

2.1 Data Source

2.1.1 Batch Data Source

The primary dataset used for this project was collected from YouTube comments using the YouTube Data API (v3). The data collection process involved programmatically searching for public YouTube videos and extracting user comments associated with those videos. The data extraction was performed using a custom script (youtube_extractor.py), which utilized a predefined list of keywords related to Malaysian topics and content. These keywords were used to filter and identify relevant videos, ensuring that the collected data aligns with the scope of the project. A sample of the extracted dataset, illustrating the structure and types of information retrieved, is shown in Figure 2.1.1.1. A list of the YouTube search topics used to guide comment collection is provided below, followed by a preview of the dataset in Figure 2.1.1.2.

	A	B	C	D	E
1	video_id	author	text	published_at	like_count
2	WZN1bJhUD60	@syedmalicksyedimam5176	INDIAN MUSLIM ARE OUR BROTHERS TOO.....IN SHA ALLAH ALLAH SWT WILL SAVE THEM.....V	2025-06-06T08:30:45Z	0
3	WZN1bJhUD60	@KarunanithiNramachandran-qw8xi	Wow !	2025-06-06T03:41:49Z	0
4	WZN1bJhUD60	@stevendawson3052	Proud for the malaysian goverment for such a decision,we all jer living happily in malaysiað	2025-06-05T16:04:34Z	2
5	WZN1bJhUD60	@miniwagen9273	All citizens of Malaysia are good people and and souls. NOT IT'S POLITICIANS.. NVR HAVE FA	2025-06-05T12:35:45Z	2
6	WZN1bJhUD60	@GloriesAlways	Actually only some minorities Extremist in the Opposition parties in Malaysia supported PAK	2025-06-05T12:13:48Z	0
7	WZN1bJhUD60	@devangerdthori3757	The FM from this country has claimed..that more people from his country die from terrorism	2025-06-05T06:19:48Z	1
8	WZN1bJhUD60	@vijayanathanstephen9308	India today ! You are propagating fake news ! The vast majority of the mulsims support Paki	2025-06-05T06:10:44Z	0
9	WZN1bJhUD60	@HussainBoa	Yes and took two victorys in UN last night	2025-06-05T02:26:35Z	0
10	WZN1bJhUD60	@LoganNathan-q8f	Good morning everyone thanks for your news congrats ðŸ‘	2025-06-05T00:06:18Z	2

Figure 2.1.1.1 : Extracted dataset sample

```
# Keyword Used
# "malaysia education system", "malaysia healthcare system", "malaysia medications", "malaysia technology", "malaysia semiconductor industry"
# "malaysia investment", "malaysia best hospital", "malaysia wild animal", "najib case in malaysia", "malaysia corruption",
# "jho low", "1mdb scandal", "malaysia military", "malaysia economy class", "malaysia protest", "malaysia mh370 case"
# "rohingya in malaysia", "zahid hamidi case", "petronas station on fire", "malaysia unemployment", "malaysia debt"
# "lim guan eng", "malaysia scam crime", "kim jong nam malaysia", "malaysia road accident crisis", "malaysia tariff response"
# "asean summit in malaysia", "macc malaysia", "missing british backpacker in malaysia", "sze fei izzuddin men's pair", "dr. M malaysia"
# "malaysian homeless records", "singapore acquire malaysian owned land", "what happened to fashionvalet", "malaysia ghost town",
# "malaysian rider championship opener", "malaysia travel", "malaysia beach", "malaysia current affairs", "malaysia opinion", "malaysia analysis"
# "malaysia issues", "Breaking news Malaysia", "klang crime", "malaysian celebrate eid", "commando malaysian"
# "sabah invicible people", "lahad datu incident", "malaysia storm", "malaysia discrimination", "malaysia unity"
# "lpg enforcement malaysia", "malaysia heart disease", "malaysia diabetes"
```

Figure 2.1.1.2 : Keyword Used

2.1.2 Real Time Data Source

The real-time data source in this project is implemented using a custom Python script that continuously streams live comments from a predefined list of Malaysia-related YouTube videos. This script utilizes the 'youtube-comment-downloader' package to retrieve new comments and yt-dlp to extract video metadata such as the title, publish date, view count, and like count. Each collected comment, enriched with video metadata, is published to the Kafka topic youtube-comments using a KafkaProducer. The data is serialized in JSON format to ensure compatibility with downstream components such as Apache Spark, which consume the data in near real-time. To avoid sending duplicates, the script maintains a set of processed comment IDs and periodically loops through the list of video IDs to check for new comments. This mechanism forms the foundation of the real-time ingestion layer within our data pipeline architecture.

Figure 2.1.2.1 shows a sample output of the extracted YouTube comment data, while Figure 2.1.2.2 displays the list of video IDs used as data sources in the pipeline.

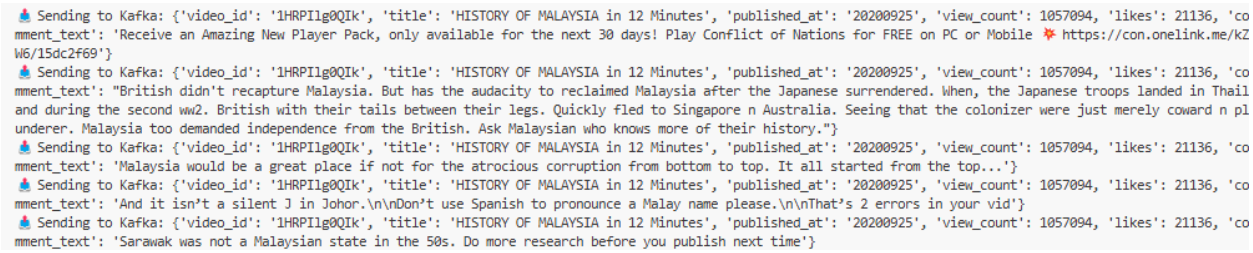

Figure 2.1.2.1 : Sample output of Data Extracted

```
VIDEO_IDS = [
    "1HRPI1g0QIk",
    "NeG9Ps0mS0Y",
    "JyzGus2bB08",
    "-Vot0XL0If8",
    "w8xPMD8ubOE",
    "g9fN4fSTS7o",
    "quaTx38WnZ8",
]
```

Figure 2.1.2.2 : Video Ids Used

2.2 Tools

The data acquisition and preprocessing pipeline relies on a set of powerful tools and libraries, each serving a specific purpose:

Extraction and Labeling Tools Used:

- YouTube Data API (v3): This official Google API is the fundamental tool for programmatically accessing YouTube data, including video information and comments. It allows for structured and controlled data extraction.
- langdetect: A Python library for language detection, used during the data acquisition phase to filter comments and retain only those identified as English, ensuring consistency for the sentiment analysis model.
- Hugging Face transformer: Hugging Face's transformers library used to assign initial sentiment labels to the scraped YouTube comments using the pre-trained 'cardiffnlp/twitter-roberta-base-sentiment' model. The pipeline was applied in batches to efficiently label the data as positive, neutral, or negative, providing a labeled dataset to be used later for training our own sentiment analysis model.
- kafka-python: The Python client for Apache Kafka, enabling the project's producer script to send collected YouTube comments as messages to a Kafka topic in real-time.
- re (Python's re module): Python's built-in regular expression module, extensively used for pattern-based text manipulation during the cleaning process, such as removing URLs and unwanted symbols.
- Apache Spark (PySpark): PySpark (the Python API for Spark) enables the application of cleaning steps and machine learning inference at scale on the streaming data from Kafka.

NLP Tools Used:

- Python's re module: Used within the clean_text function for regular expression-based text manipulation, such as removing URLs and unwanted symbols.
- joblib: This library is used to load the pre-trained tfidf_vectorizer.pkl and best_sentiment_model.pkl. These models were likely created using other Python machine learning libraries (e.g., scikit-learn), which in turn might leverage components from NLTK or spaCy for their underlying text processing capabilities during model training.

- Apache Spark (PySpark): Spark's powerful distributed processing capabilities are utilized to apply these cleaning and sentiment prediction steps at scale on streaming data. User Defined Functions (UDFs) like clean_text_udf and predict_udf allow custom Python logic to be executed efficiently across the Spark cluster.

2.3 Data Cleaning

The data preprocessing in this project is primarily handled within the spark_stream.py script, which processes real-time data from Kafka.

Text Cleaning Steps:

- **Lowercasing:** The clean_text function explicitly converts all comment text to lowercase using text.lower(). This standardizes the text, ensuring that words like "Happy" and "happy" are treated the same.
- **Punctuation Stripping/Symbol Removal:** The clean_text function utilizes regular expressions to remove unwanted characters. It removes URLs (re.sub(r"http\S+|www\S+", '', text)) and other non-alphanumeric symbols or emojis (re.sub(r'[^a-zA-Z0-9\s.,!?]', '', text)). This helps in reducing noise and focusing on meaningful text content.
- **Whitespace Normalization:** Redundant spaces are removed and normalized using re.sub(r'\s+', ' ', text).strip() to ensure consistent spacing.
- **Tokenization, Stopwords Removal, Lemmatization/Stemming:** While not explicitly coded as separate functions in spark_stream.py, these critical steps are implicitly handled by the pre-trained tfidf_vectorizer.pkl. A TF-IDF vectorizer typically performs tokenization (breaking text into words), can be configured to remove common stopwords (e.g., "the," "is"), and might incorporate stemming or lemmatization during its training phase to reduce words to their base forms, thereby preparing the text for machine learning models.

3.0 Sentiment Model Development

This section describes the structured approach to developing, training, and evaluating machine learning models for sentiment analysis of YouTube comments, focusing on metrics such as accuracy and F1 score. A comparative methodology was used to determine the most effective model and feature extraction technique, ensuring optimal performance in terms of accurate, efficient, and scalable real-time sentiment classification.

3.1 Model Choice

Our group had selected two supervised machine learning models to classify the sentiment results into ‘Positive’, ‘Negative’, and ‘Neutral’. These models were chosen due to their interpretability, efficiency, and proven effectiveness in text classification tasks. The chosen models were Logistic Regression and Naive Bayes.

- **Logistic Regression**

  It offers simplicity, strong performance, and transparent decision-making, making it easy to interpret the importance of individual features. This model's ability to provide clear insights into how different features contribute to sentiment classification makes it a valuable tool for text analysis.

- **Naive Bayes**

  Specifically the Multinomial Naive Bayes variant, it is highly efficient for text classification tasks. It excels in handling high-dimensional data, such as text, and offers robust performance, particularly when dealing with large datasets. The combination of these two models allowed for a comprehensive comparison to identify the most effective model for accurately predicting sentiment in YouTube comments.

3.2 Training Process

The training process for the sentiment analysis model involved several key steps, from data labeling and preprocessing to model training and evaluation. Below is a detailed breakdown of each step.

3.2.1 Data Labeling

Before model development, the extracted YouTube comments were labeled as 'Positive', 'Neutral', or 'Negative' using Hugging Face's Transformers library. The pre-trained 'cardiffnlp/twitter-roberta-base-sentiment' model was used for this task. To efficiently label a large volume of comments, batch processing was employed, ensuring a high-quality labeled dataset for model training.

3.2.2 Data Preprocessing

Once the comments were labeled, they were preprocessed to remove unnecessary elements and ensure they were in a clean and consistent format. This step included the following:

- Text Cleaning: Removal of URLs, hashtags, user mentions, and punctuation.
- Text Normalization: Converting all text to lowercase and eliminating extra whitespace.
- Handling Missing Data: Ensuring there were no missing or invalid entries in the dataset.
- Text Length Filtering: Comments with fewer than three characters were discarded to remove irrelevant data.

3.2.3 Data Splitting

After preprocessing, the data was divided into training and testing sets using an 80-20 split as illustrated in Figure 3.2.1. This means that 80% of the dataset was used to train the models, while 20% was held back for evaluation. This split ensured that the models were trained on a sufficient amount of data and evaluated on an unbiased, unseen subset.

```
def split_data(self, test_size=0.2, random_state=42):
    """Split data into training and testing sets"""
    logger.info("Splitting data into train and test sets...")

    X = self.df['text']
    y = self.df['label']

    self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(
        X, y, test_size=test_size, random_state=random_state, stratify=y
    )

    logger.info(f"Training set size: {len(self.X_train)}")
    logger.info(f"Test set size: {len(self.X_test)}")
```

Figure 3.2.1 8/2 Data Splitting

3.2.4 Model Initialization

In Figure 3.2.2, two machine learning models were selected for training:

- Logistic Regression: Known for its simplicity, efficiency, and effectiveness in text classification tasks.

- Naive Bayes: Specifically, the Multinomial Naive Bayes classifier was chosen, which is well-suited for text data and performs well in high-dimensional feature spaces.

```
def initialize_models(self):
    """Initialize the two most suitable machine learning models for sentiment analysis"""
    logger.info("Initializing models...")

    self.models = {
        'Logistic Regression': LogisticRegression(random_state=42, max_iter=1000),
        'Naive Bayes': MultinomialNB()
    }

    logger.info(f"Initialized {len(self.models)} models: {list(self.models.keys())}")
```

Figure 3.2.2 Model Initialization

3.2.5 Pipeline Creation

A machine learning pipeline was created for each model to streamline the process of transforming the text data and applying the model. As illustrated in Figure 3.2.3, the pipeline consisted of:

- TF-IDF Vectorizer: This was used to transform the raw text into numerical features. The TF-IDF approach ensures that important words are given more weight, while less informative terms are down-weighted.

- Model Fitting: The pipeline incorporated the chosen machine learning model (either Logistic Regression or Naive Bayes) to train the model on the preprocessed text data.

```
for name, model in self.models.items():
    logger.info(f"Training {name}...")

    # Create pipeline with TF-IDF vectorizer
    pipeline = Pipeline([
        ('tfidf', TfidfVectorizer(
            stop_words='english',
            max_features=10000,
            ngram_range=(1, 2),
            min_df=2,
            max_df=0.95
        )),
        ('classifier', model)
    ])
```

Figure 3.2.3 Model Pipeline with TF-IDF Vectorizer

3.2.6 Model Training

Both models were trained on the training dataset, and the training process consisted of the following key steps:

- Model Fitting: Each model was trained using the training data, with the TF-IDF vectorizer applied to convert the text data into numerical feature vectors. This transformation allowed the models to learn from the textual content and extract meaningful patterns for sentiment classification.

- Performance Evaluation: Upon completion of the training, the models were evaluated on the test dataset to assess their performance. The evaluation focused on the models' ability to accurately classify YouTube comments into the predefined sentiment categories: 'Positive', 'Neutral', or 'Negative'.

3.2.7 Model Saving

After training and evaluating both models, the best-performing model was selected based on its performance metrics (particularly accuracy and F1-score). This model was chosen for future deployment in the sentiment analysis pipeline.

The following Figure 3.2.4 also shows the code for both selection and saving of models, along with the TF-IDF vectorizer, was saved for future use. This ensures that the trained model can be easily applied to new data for real-time sentiment classification. Additionally, the model metadata (e.g., accuracy, F1-score, and training time) was saved for reference.

```
def save_best_model(self):
    """Save the best performing model"""
    best_idx = np.argmax(self.results['Accuracy'])
    best_model_name = self.results['Model'][best_idx]
    best_pipeline = self.trained_pipelines[best_model_name]

    # Save the best model
    model_path = f"model/best_sentiment_model_{best_model_name.replace(' ', '_').lower()}.pkl"
    joblib.dump(best_pipeline, model_path)

    # Extract and save the TF-IDF vectorizer separately for Spark streaming
    tfidf_vectorizer = best_pipeline.named_steps['tfidf']
    vectorizer_path = "model/tfidf_vectorizer.pkl"
    joblib.dump(tfidf_vectorizer, vectorizer_path)

    # Extract and save the classifier separately
    classifier = best_pipeline.named_steps['classifier']
    classifier_path = f"model/{best_model_name.replace(' ', '_').lower()}_classifier.pkl"
    joblib.dump(classifier, classifier_path)
```

Figure 3.2.4 Best Model Selection and Saving

3.3 Model Evaluation

After training the sentiment analysis models, it was crucial to evaluate their performance to determine which model performed best for classifying YouTube comments as 'Positive', 'Neutral', or 'Negative'. This evaluation involved measuring key metrics such as accuracy, precision, recall, F1-score, and cross-validation scores. Additionally, confusion matrices were generated to visually assess the model's predictions. The following steps outlined the model evaluation process.

3.3.1 Performance Metrics

To assess the performance of both the Logistic Regression and Naive Bayes models, several key metrics were computed with the codes shown in Figure 3.3.1:

- Accuracy: This measures the proportion of correctly predicted labels out of the total predictions. It provides an overall measure of model performance.

- Precision: This metric indicates the proportion of correctly predicted positive observations relative to all predicted positive observations. It is particularly important in cases where the cost of false positives is high.

- Recall: This metric measures the proportion of correctly predicted positive observations relative to all actual positive observations. High recall indicates that the model successfully identifies most of the true positives.

- F1-Score: The F1-score is the harmonic mean of precision and recall, providing a balanced measure of the model's performance when dealing with imbalanced data. It is particularly useful when both precision and recall are important.

```
# Calculate metrics
accuracy = accuracy_score(self.y_test, y_pred)
precision = precision_score(self.y_test, y_pred, average='weighted')
recall = recall_score(self.y_test, y_pred, average='weighted')
f1 = f1_score(self.y_test, y_pred, average='weighted')
```

Figure 3.3.1 Metric Calculations

3.3.2 Cross-Validation

To assess the robustness and generalizability of the models, cross-validation was performed as shown in Figure 3.3.2. Cross-validation helps mitigate the risk of overfitting by testing the model's performance on different subsets of the training data.

- Cross-Validation Accuracy: The average accuracy from 5-fold cross-validation was computed to evaluate the model's stability across different training sets.

```
# Cross-validation
cv_scores = cross_val_score(pipeline, self.X_train, self.y_train, cv=5, scoring='accuracy')
cv_accuracy = cv_scores.mean()
```

Figure 3.3.2 Cross-Validation Accuracy

3.3.3 Confusion Matrix

A confusion matrix was generated for both models to visually assess their performance. The confusion matrix shows the true positives, false positives, true negatives, and false negatives, which helps to identify the types of classification errors the model is making. This was represented as a heatmap for clearer visualization. The following Figure 3.3.3.1 and Figure 3.3.3.2 show the coding of the creation of confusion matrix and the heatmap visualization of confusion matrix for each model (Logistics Regression and Naive Bayes) respectively.

```python
def create_confusion_matrices(self):
    """Create confusion matrix visualization for both models"""
    logger.info("Creating confusion matrices visualization...")

    # Get unique labels for consistency
    unique_labels = sorted(self.df['label'].unique())

    # Set up the figure with subplots
    fig, axes = plt.subplots(1, 2, figsize=(15, 6))
    fig.suptitle('Confusion Matrices - Model Performance Comparison', fontsize=16, fontweight='bold')

    for idx, (name, pipeline) in enumerate(self.trained_pipelines.items()):
        # Get predictions
        y_pred = pipeline.predict(self.X_test)

        # Create confusion matrix
        cm = confusion_matrix(self.y_test, y_pred, labels=unique_labels)

        # Create heatmap
        ax = axes[idx]
        sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                    xticklabels=unique_labels, yticklabels=unique_labels,
                    ax=ax, cbar=True)
        ax.set_title(f'{name}', fontsize=14, fontweight='bold')
        ax.set_xlabel('Predicted Label')
        ax.set_ylabel('True Label')

        # Add accuracy score to the plot
        accuracy = accuracy_score(self.y_test, y_pred)
        ax.text(0.5, -0.1, f'Accuracy: {accuracy:.3f}',
                transform=ax.transAxes, ha='center', fontweight='bold')

    # Adjust layout and save
    plt.tight_layout()
    plt.savefig('model/confusion_matrices.png', dpi=300, bbox_inches='tight')
    plt.close()
```

Figure 3.3.3.1 Confusion Matrix Coding

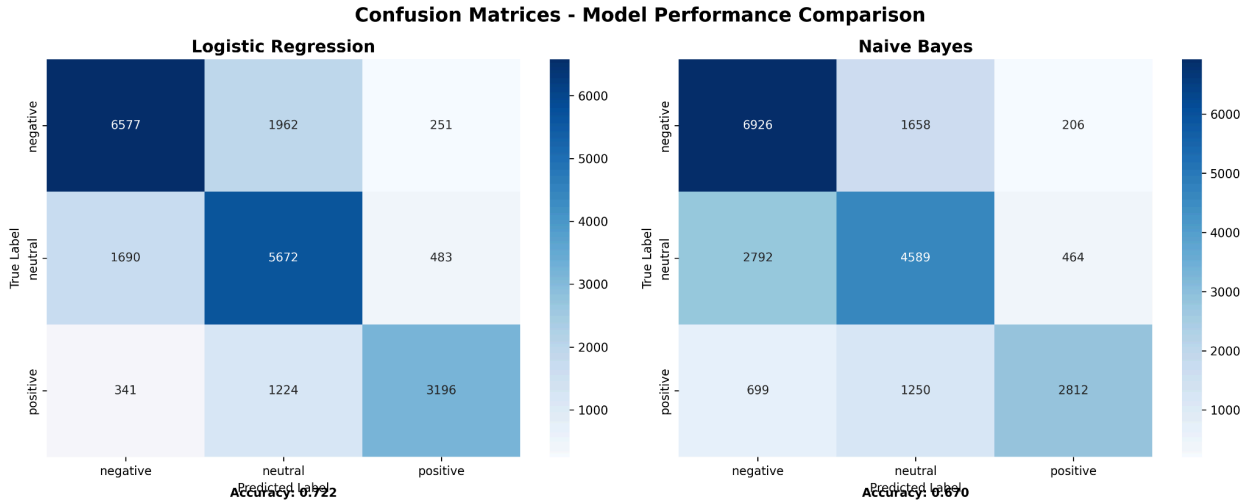

Figure 3.3.3.2 Model Confusion Matrix Heatmap

3.3.4 Final Model Selection

Based on the evaluation metrics as shown in Figure 3.3.4, which is the comparison of model performance, the Logistic Regression model was selected as the best-performing model for sentiment analysis. This model achieved the highest accuracy, along with superior precision, recall, and F1-score when compared to the Naive Bayes model. These metrics were key factors in selecting Logistic Regression for deployment.

The Logistic Regression model, along with its associated TF-IDF vectorizer and classifier, was then saved for future use in real-time sentiment classification. This ensures that the trained model can be applied to new data with confidence in its performance.

```
DETAILED MODEL COMPARISON RESULTS
==================================


  Accuracy:        0.7219
  Precision:       0.7296
  Recall:          0.7219
  F1-Score:        0.7233
  CV Accuracy:     0.7147
  Training Time:   28.2555s
  Prediction Time: 0.5654s


  Accuracy:        0.6696
  Precision:       0.6773
  Recall:          0.6696
  F1-Score:        0.6674
  CV Accuracy:     0.6680
  Training Time:   4.9019s
  Prediction Time: 0.5742s
```

Figure 3.3.4 Model Comparison Results

4.0 Apache System Architecture

This section describes the implementation of a sentiment analysis pipeline for YouTube comments. The process includes model training, data ingestion via a Python scraper and Kafka, and real-time processing using Apache Spark. The sentiment-labeled comments are indexed in Elasticsearch for efficient search and visualized with Kibana. The entire pipeline is containerized with Docker for scalability and reproducibility, with key components like Apache ZooKeeper, Kafka, Elasticsearch, and Spark working together to deliver real-time sentiment analysis.

4.1 Sentiment Analysis Workflow

According to Figure 4.1, the sentiment analysis pipeline for YouTube comments proceeds as follows:

Model Training

First, we collect a corpus of approximately 100000 comments from Malaysia-related YouTube videos and label them using a Hugging Face transformer. This labeled dataset is exported to CSV and fed into a Spark MLlib training job. In Spark, we perform text preprocessing (cleaning, tokenization, stop-word removal, lemmatization), fit a TF-IDF vectorizer, and train several classifiers (Logistic Regression, Naive Bayes, etc.). After validating on a held-out set, we select the best TF-IDF transformer (saved as tfidf_vectorizer.pkl) and classifier (saved as best_model.pkl) and store these artifacts in a shared model volume for production use.

Data Ingestion and Kafka Buffering

At runtime, a Python scraper continuously polls the YouTube API for new public comments. Each comment, together with its metadata (video ID, author, timestamp, like count), is serialized into a JSON object and published to the Kafka topic named youtube-comments. Apache ZooKeeper manages the cluster's metadata and consumer offsets, while Kafka provides a durable, replayable buffer that decouples comment production from downstream consumers.

Real-Time Stream Processing with Spark

An Apache Spark Structured Streaming application subscribes to the youtube-comments topic

and ingests the JSON stream in micro-batches. Spark applies the same preprocessing steps used offline (cleaning, tokenization, stop-word removal, lemmatization), then transforms each comment into a TF-IDF vector using the pre-trained tfidf_vectorizer.pkl. The broadcasted best_model.pkl classifier predicts positive, negative, or neutral sentiment for each comment. Spark enriches each JSON record with the cleaned text and sentiment label before forwarding it.

Indexing and Visualization

Finally, Spark writes the sentiment-annotated records to an Elasticsearch index. Elasticsearch shards and replicates the data for fast full-text search and aggregation. Kibana connects to this index and renders an interactive dashboard featuring pie charts of sentiment distribution, time-series graphs tracking sentiment trends by video or channel, and filters for detailed exploration. This step-by-step workflow, managed via Docker Compose, ensures end-to-end reproducibility and scalability.

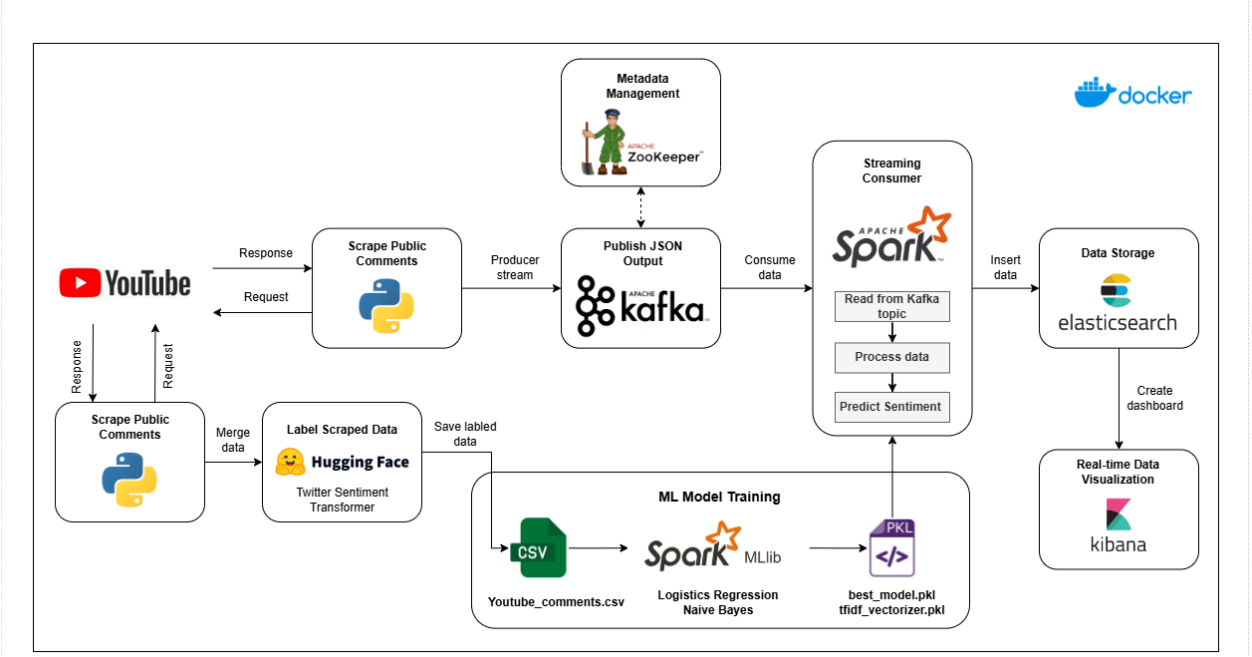

Figure 4.1 YouTube Sentiment Analysis Pipeline

4.2 Component Roles and Configuration

In this project, our group has used Docker and Docker Compose to containerize and orchestrate all components of the YouTube sentiment analysis pipeline. Below is a breakdown of each service defined in the project's docker-compose.yml as shown in Appendix D, along with the custom model-training image defined in Dockerfile.model as shown in Appendix E. Figure 4.2.1 below shows all the images and containers managed by Docker Desktop.

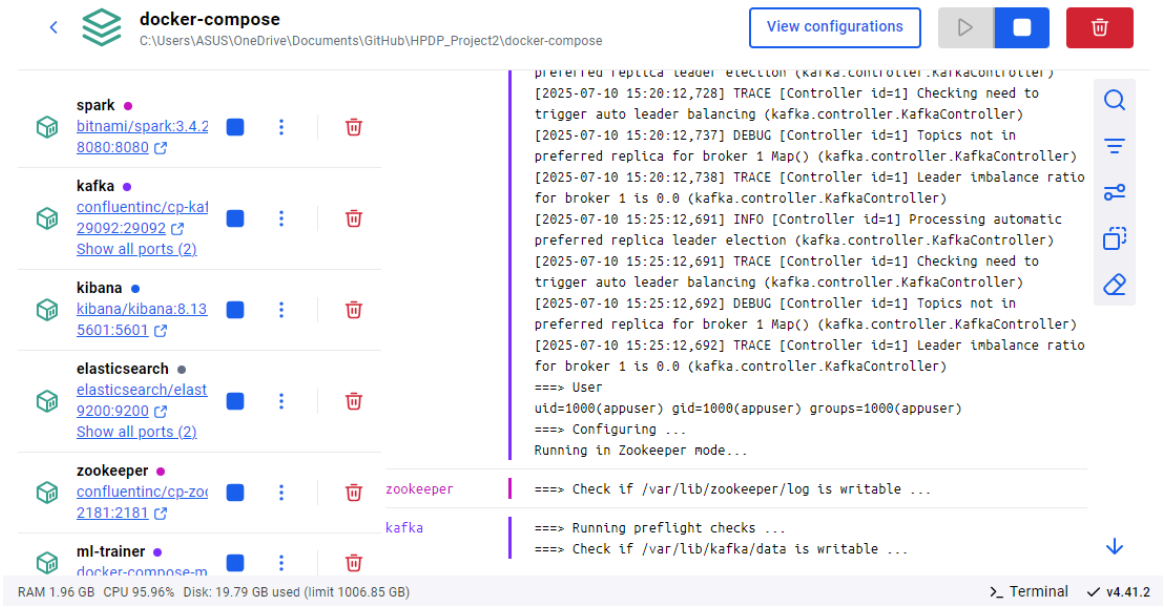

Figure 4.2.1 Docker Desktop

Apache ZooKeeper

- Image: confluentinc/cp-zookeeper:7.6.1
- Role: Acts as the centralized coordination service for the Kafka cluster, managing broker metadata, leader election, and consumer offsets.
- Configuration Highlights:
    - ZOOKEEPER_CLIENT_PORT: 2181
    - ZOOKEEPER_TICK_TIME: 2000
    - Port 2181 bound for both intra-cluster and client connections.

Apache Kafka

- Image: confluentinc/cp-kafka:7.6.1
- Role: Serves as the high-throughput, distributed messaging backbone, buffering incoming YouTube comments and decoupling producers from consumers.
- Configuration Highlights:
    - Depends on the zookeeper service for cluster coordination.
    - Exposes 29092 for inter-container traffic and 9092 on the host.
    - KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://kafka:29092,PLAINTEXT_HOST://localhost:9092 enables both container-to-container and host-accessible endpoints.
    - KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR=1 for a single-node development setup.

Elasticsearch

- Image: docker.elastic.co/elasticsearch/elasticsearch:8.13.4
- Role: Stores and indexes enriched comment documents (original text, cleaned text, predicted sentiment, metadata) to support fast full-text search, aggregations, and time-series queries.
- Configuration Highlights:
    - Single-node mode (discovery.type=single-node) with security disabled (xpack.security.enabled=false).
    - JVM heap capped at 1 GB (ES_JAVA_OPTS=-Xms1g -Xmx1g).
    - Exposes 9200 (REST) and 9300 (transport) ports.
    - bootstrap.memory_lock=true plus unlimited memlock limits to prevent swapping.

Figure 4.2.2 illustrates the data indexing process of the Elasticsearch index stored in the youtube-comments1-index.

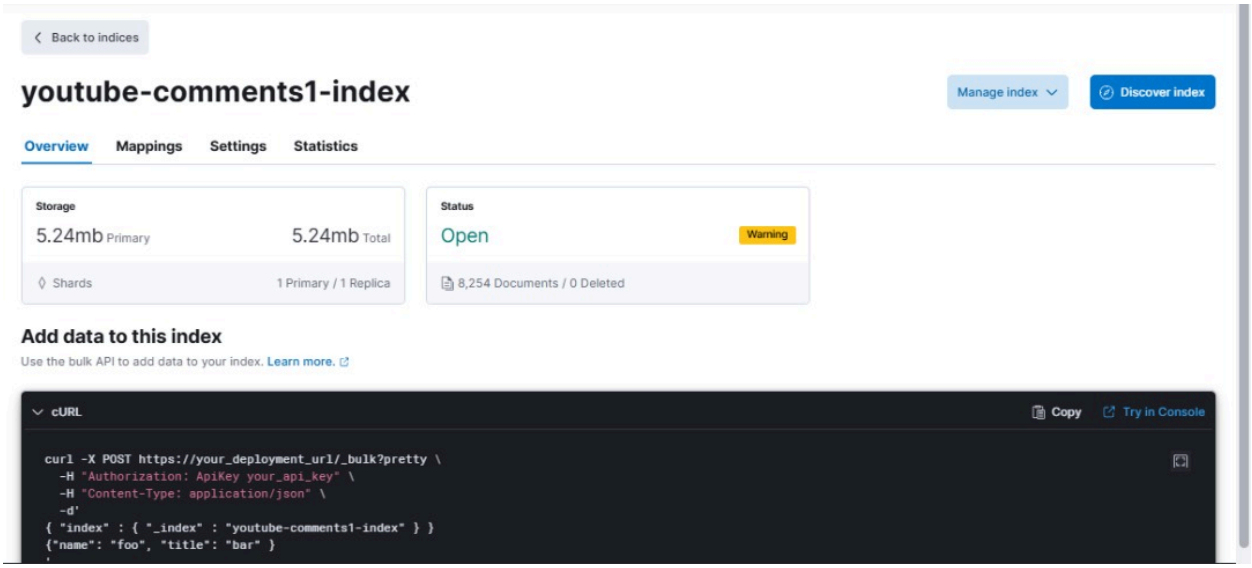

Figure 4.2.2 youtube-comments1-index

Kibana

- Image: docker.elastic.co/kibana/kibana:8.13.4

- Role: Provides the visualization layer for interactive dashboards, connecting to Elasticsearch to render real-time sentiment metrics.
- Configuration Highlights:
    - Depends on elasticsearch to ensure the data store is available.
    - Exposes port 5601 for the web UI.
    - Configured via ELASTICSEARCH_HOSTS=http://elasticsearch:9200.

Apache Spark

- Image: bitnami/spark:3.4.2
- Role: The structured-streaming engine that consumes raw comments from Kafka, executes NLP preprocessing and model inference, then writes enriched records to Elasticsearch.
- Configuration Highlights:
    - Depends on both kafka and elasticsearch for end-to-end connectivity.
    - Exposes Spark master UI on port 8080.

- Environment variables: SPARK_MODE=master, SPARK_MASTER_HOST=spark, SPARK_MASTER_PORT=7077

ML Trainer

- Build Context: Defined by Dockerfile.model, producing an image for offline model training and artifact generation.
- Role: Periodically retrains sentiment classifiers, serializes the top-performing vectorizer and model, and writes them to the shared ./model volume for Spark to consume.
- Configuration Highlights:
  - Depends on kafka (for fresh data ingestion) and elasticsearch (for storing evaluation metrics).
  - Environment: PYTHONUNBUFFERED=1 to stream logs in real time.

5.0 Analysis & Results

This section presents the analysis and key findings from the real-time sentiment analysis of YouTube comments. Using Elasticsearch for data indexing and storage, and Kibana for creating interactive dashboards, the analysis focused on sentiment distribution, trends over time, and category breakdowns. This approach allowed for efficient exploration of the data, uncovering important patterns and insights.

5.1 Visualisation

Figure 5.1 shows the Real-time YouTube Sentiment Analysis Dashboard built in Kibana capturing data from 4:23 AM to 4:33 AM on 8th July 2025 within 5 YouTube videos about the history and politics of Malaysia.

Figure 5.1 Real-time YouTube Sentiment Analysis Kibana Dashboard

Real-Time Sentiment Timeline

The donut chart in Figure 5.1.1 displays the sentiment distribution of YouTube comments related to Malaysian content. The visualization reveals a striking imbalance, with negative sentiment dominating at 98.51%. This indicates that the overwhelming majority of comments analyzed expressed criticism, dissatisfaction, or unfavorable views. In contrast, positive and neutral sentiments together make up only a small fraction of the responses, barely visible in the chart. Such a disproportionate sentiment trend suggests a highly critical or emotionally charged

response from viewers toward the analyzed videos. This could be attributed to the nature of the content, viewer biases, or even flaws in sentiment detection accuracy. Overall, the chart highlights a strong lean toward negative engagement within the comment section of the selected Malaysian YouTube videos.

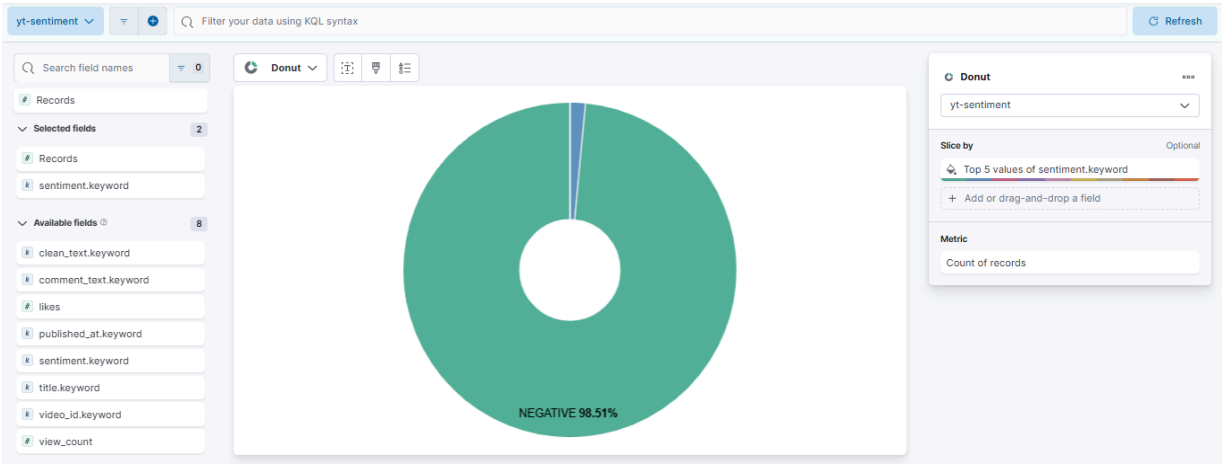

Figure 5.1.1 : Pie chart sentiment percentage

Batch Historical Trends

The bar chart Figure 5.1.2 illustrates the total number of likes received by YouTube videos related to Malaysia, grouped by their publish month from July 2024 to July 2025. The data reveals that February 2025 saw a notable increase in engagement, accumulating over 1.5 million likes, suggesting that one or more popular videos were published during that period. The highest level of engagement occurred in June 2025, with nearly 4 million likes, indicating a significant surge in audience interaction. This spike could be attributed to the release of a viral video, the publication of several high-performing videos, or content that aligned closely with trending or highly relevant topics at the time. In contrast, the other months namely from July 2024 to January 2025, as well as March to May 2025 recorded negligible or zero likes, which may imply that no new content was published during those periods or that the videos released failed to attract much viewer interest or interaction.

Figure 5.1.2 : Monthly Total Likes for Malaysia YouTube Content

Sentiment Contribution by Video

The treemap in Figure 5.1.3 illustrates the distribution of YouTube video popularity related to Malaysian topics, using the median view count as the metric. Each block represents a specific video, with the size and percentage indicating its relative contribution to overall viewership among the top five titles. The video titled "HISTORY OF MALAYSIA in 12 Minutes" stands out prominently, accounting for 47.33% of the total median views, reflecting its broad appeal and high engagement. In second place, "Sejarah Malaysia - Berapa Lama Tanah Melayu Dijajah?" contributes 26.8%, showing significant interest in local historical narratives. The video "Foreigners are RUSHING into Malaysia. Here's Why." follows with 14.66%, suggesting that topics involving immigration and global perception resonate with viewers. Meanwhile, "Why the US & China both Need Malaysia" holds 8.79%, indicating moderate attention toward Malaysia's geopolitical importance. Lastly, "A Super Quick History of Malaysia" comprises just 2.42%, perhaps due to competition from similar content or a shorter format. This visualization emphasizes that educational and historical videos, especially those offering concise summaries garner the most viewer interest.

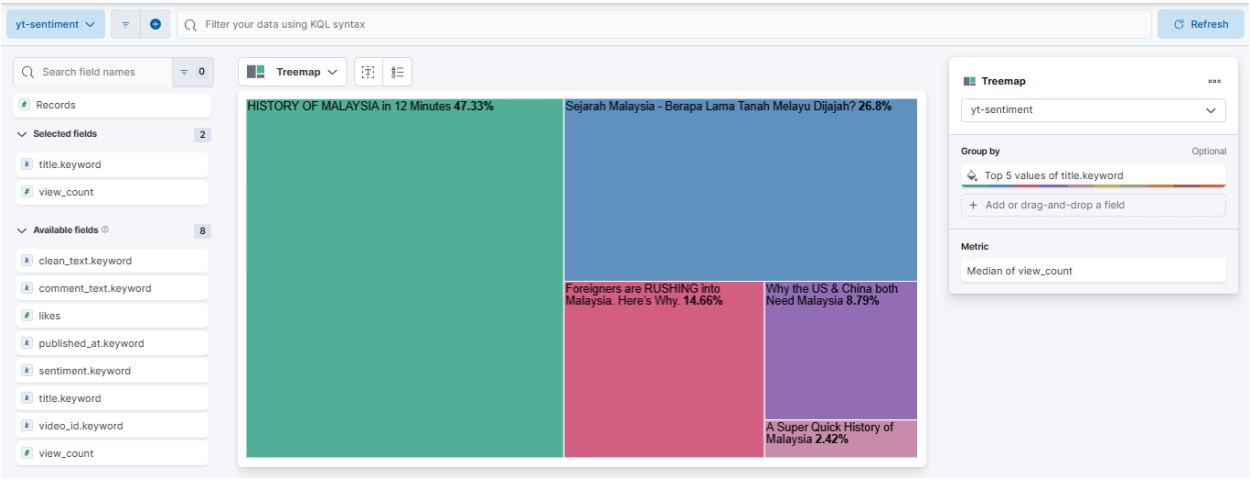

Figure 5.1.3 : Treemap represents the distribution of video popularity or view contribution among several Malaysia-related YouTube videos

Audience Engagement: Likes per YouTube Video

The bar chart figure 5.1.4 illustrates the maximum number of likes received by several Malaysian-themed YouTube videos, offering insight into viewer engagement across different content types. The video titled "HISTORY OF MALAYSIA in 12 Minutes" stands out

significantly, garnering over 20,000 likes, which highlights its popularity and the strong audience interest in historical content presented in a concise format. Following that, "Why the US & China both Need Malaysia" attracted approximately 12,000 likes, indicating substantial viewer attention towards geopolitically themed videos. Two other videos, likely including "Foreigners are RUSHING into Malaysia" and "A Super Quick History of Malaysia", received between 5,000 and 7,000 likes, showing a moderate level of engagement. Meanwhile, the final video recorded fewer than 3,000 likes, suggesting comparatively lower viewer interest or visibility. Overall, the chart emphasizes that videos addressing Malaysia's history and global relevance tend to generate higher levels of interaction from viewers.

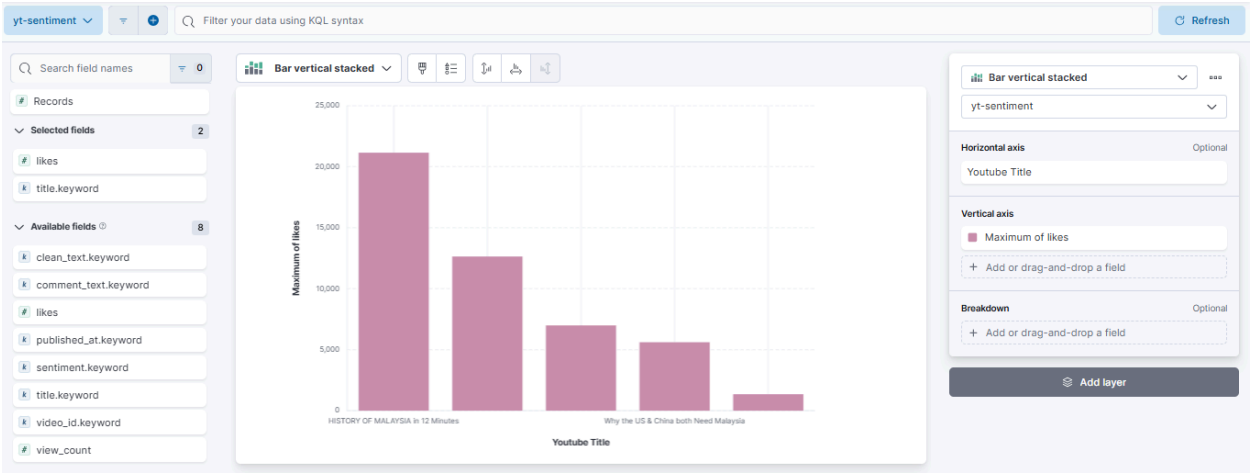

Figure 5.1.4: Bar chart illustrates the maximum number of likes received by several Malaysian-themed YouTube video

Total Comment Count

Figure 5.1.5 presents a "Total Youtube Comments" count of 7,181. The selection of comment_text_keyword as a "Selected field" indicates that this total specifically represents comments that contain predefined keywords relevant to the analysis. Therefore, the 7,181 comments reflect a targeted subset of the overall YouTube comment data, filtered to focus on specific textual content pertinent to the ongoing YouTube sentiment study.

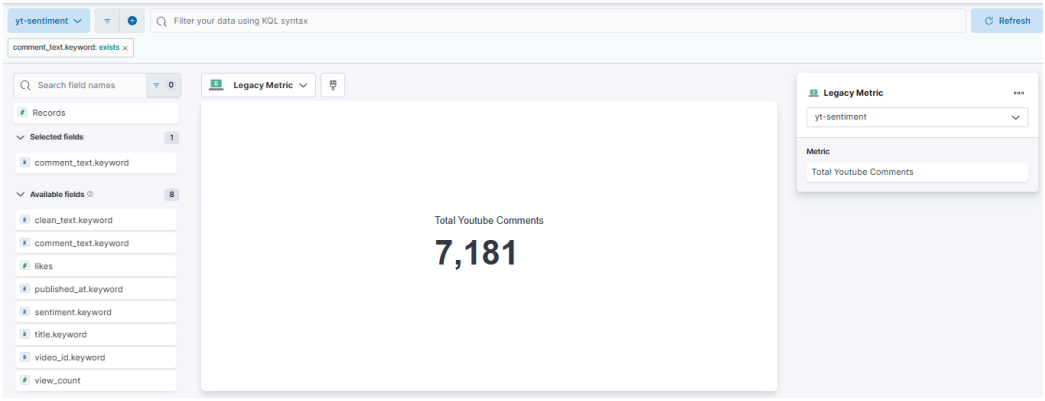

Figure 5.1.5 : Total number of YouTube comments

Word Cloud

The word cloud Figure 5.1.6 visualization highlights the most frequently occurring phrases and keywords extracted from YouTube comment data related to Malaysian content. The dominant term “Other” likely represents a grouping of miscellaneous or uncategorized comments, but surrounding it are several notable and context-rich phrases. Phrases such as "1957 Malaysia merdeka," "16 September 1963 lah kan?" and "brunei wrongly illustrated…” reflect a strong emphasis on historical and geopolitical discourse, with many users referencing Malaysia’s independence dates and regional complexities. Additionally, comments like "we need to change the Malay!" and "Malaysia will get a better leader since 2020 is insane. Well done" suggest political and social reflections by viewers. The presence of phrases such as "the intro is so exciting... greetings from Germany" and "thank you for your video masterpieces" indicates a portion of international viewers engaging positively. Overall, the keyword distribution provides insights into viewer sentiment, ranging from historical interest and national pride to political critique and global engagement.

Figure 5.1.6 : Word cloud visualization highlights the most frequently occurring phrases

Total likes sum of video

Figure 5.1.7 shows the total number of likes gathered from Malaysia-related YouTube videos. Altogether, these videos received 112,637,424 likes, which highlights just how much people are engaging with this type of content. The large number suggests that viewers are highly interested in topics related to Malaysia, including its history, politics, and culture. This figure was generated using Kibana's Legacy Metric tool, which pulls together all the like counts from the dataset into a single, easy-to-read summary.

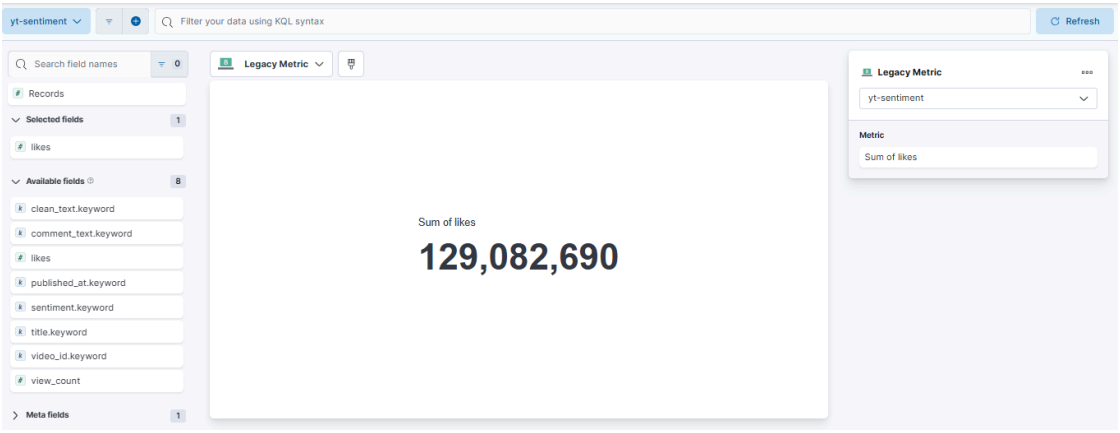

Figure 5.1.7 : Cumulative number of likes

5.2 Findings and Insights

Based on the sentiment analysis and the visualizations from the dataset, several observations can be made about how viewers respond to Malaysia-related YouTube content. The results clearly show that most of the comments carry a negative tone. A large portion of the comments, around 98.51 percent, were classified as negative. This suggests that viewers reacted critically or were not satisfied with the content. The high percentage might be due to certain videos touching on sensitive topics that trigger strong emotional responses from the audience.

On the other hand, very few comments were found to be positive or neutral. Both categories together made up less than two percent of the total comments. This shows that most people felt strongly about what they watched and were more likely to share critical opinions rather than give neutral or supportive feedback.

Some of the comments included phrases such as "brunei wrongly illustrated" or statements involving political and religious issues. These examples reflect how certain topics in the videos can lead to disagreement or even backlash. From this, it can be seen that controversial or misunderstood content tends to receive more negative attention from viewers.

In terms of video popularity, the results show that short and informative videos performed the best. The video titled *"HISTORY OF MALAYSIA in 12 Minutes"* received the highest number of likes and accounted for the majority of total engagement. Viewers also showed high interest in videos discussing Malaysia's role in global politics, such as *"Why the US & China both Need Malaysia"*. This shows that topics involving Malaysia's history and international importance tend to attract more attention.

The analysis also recorded a total of 129,082,690 likes across all videos, which shows a high level of engagement from the audience. This number reflects that Malaysian-related content on YouTube is widely viewed and interacted with. Some comments even came from international viewers, as seen in phrases like "thank you for your video masterpieces, greetings from Germany", which suggests that the content has also reached a global audience.

Looking at monthly trends, there were clear spikes in viewer interaction. In February and June 2025, the number of likes increased sharply. This might be because of the release of popular or viral videos during those periods. In contrast, some months such as July 2024 to January 2025 had very low engagement, which could be due to fewer uploads or less impactful content.

In conclusion, the sentiment and engagement analysis shows that Malaysian YouTube content that is educational, historical, or geopolitically relevant tends to perform well. Viewer sentiment is mostly positive, though sensitive topics can lead to negative feedback. Overall, the insights gathered help us understand how viewers interact with such content over time.

6.0 Optimization & Comparison

This section discusses the optimization techniques applied to enhance the performance of the YouTube comments sentiment analysis pipeline and compare the operational characteristics of batch and streaming processing modes with sentiment model comparison.

6.1 Optimization strategies

To enhance the overall performance and reliability of our sentiment analysis pipeline, several targeted optimizations were implemented across different layers of the system. These improvements focused on optimizing deployment, scalability, data processing, and real-time capabilities. The optimizations are discussed in detail below, organized by key strategies.

6.1.1 Containerization with Docker & Docker Compose

For seamless deployment and enhanced scalability, all components of the sentiment analysis pipeline, including Kafka, Spark, Elasticsearch, and Kibana, were containerized using Docker. This allowed for isolated resource allocation for each component, ensuring that dependencies and configurations did not conflict between services. Docker Compose was used to manage the orchestration of these containers, simplifying the setup, scaling, and maintenance of the system. This containerization approach provided flexibility in managing resources, simplified the deployment process, and ensured consistent environments across development, testing, and production stages. The use of Docker and Docker Compose also facilitated quick scaling, as additional containers could be spun up easily to meet increased demand.

6.1.2 Distributed Processing with Apache Spark

To optimize the processing layer, Apache Spark was configured to run in a distributed manner. The system was optimized to process data in 10-second micro-batches, striking a balance between throughput and latency. This configuration ensured that the system could handle a continuous data stream efficiently without overwhelming resources. Additionally, the sentiment model and TF-IDF vectorizer were kept in memory throughout the processing pipeline to avoid the inefficiency of repeated loading, which would otherwise degrade performance. This optimization allowed Spark to quickly perform model inference without the delays typically

associated with disk I/O. By caching essential components and fine-tuning Spark's processing parameters, we achieved stable performance and minimized latency in real-time processing.

6.1.3 Real-Time Data Handling with Apache Kafka

For handling real-time data ingestion, Apache Kafka played a pivotal role in ensuring that YouTube comments were efficiently streamed into the processing pipeline. Kafka was configured with multiple consumer groups and optimized polling settings to maximize the rate at which comments were ingested. This setup allowed for better parallelization and increased throughput, ensuring that the system could handle a high volume of incoming comments with minimal delay. Kafka's durability and scalability features ensured that data could be reliably processed in real-time, enabling the continuous flow of information to the Spark processing layer without interruptions.

6.2 Comparison

6.2.1 Batch vs Real-Time Streaming Processing

The sentiment analysis system implemented two distinct processing paradigms, each optimized for different phases of the pipeline. Batch processing was utilized during the model development phase, where the entire historical dataset, comprising approximately 100,000 YouTube comments, was processed as a single batch. This approach allowed for comprehensive model training and evaluation, with the full dataset being cleaned, vectorized, and passed through the classification algorithms in a single operation. Batch processing generated static outputs, including serialized model files, evaluation metrics, and test predictions, which were used to establish the model's baseline performance, achieving an accuracy of 72.2% using the Logistic Regression model.

On the other hand, the real-time processing system was designed for operational deployment, where individual comments were processed as they were ingested through Kafka. Using Spark's Structured Streaming API, the system processed incoming comments in micro-batches every 10 seconds. This real-time pipeline applied the same preprocessing and classification steps as the batch process but on a continuous data stream. It maintained sub-second latency for processing individual comments while dynamically updating both the Elasticsearch index and Kibana

visualizations. The real-time pipeline exhibited performance comparable to the batch-trained model, although it shared similar limitations when dealing with nuanced language and sentiment bias, especially when processing live data.

Both batch and real-time processing used the same core model and preprocessing logic, yet they diverged in their data handling characteristics and operational requirements. Batch processing favored completeness, offering a thorough analysis of the full dataset, whereas real-time processing emphasized responsiveness, enabling near-instantaneous updates to the system. These complementary approaches ensured that the system could deliver both high-quality, complete analyses and fast, real-time sentiment insights, tailored to the specific needs of the deployment.

6.2.2 Sentiment Model Comparison

Figure 6.2.2 presents a comprehensive comparison between the Logistic Regression and Naive Bayes models based on key evaluation metrics: accuracy, precision, recall, F1-score, and training time. The visual comparison reveals significant differences in the performance of the two models.

Logistic Regression outperformed Naive Bayes across all key metrics, achieving an accuracy of 0.722, compared to 0.670 for Naive Bayes. This indicates that Logistic Regression was more effective at classifying YouTube comments into the correct sentiment categories, 'Positive', 'Neutral', and 'Negative'. In terms of precision, which measures the accuracy of positive sentiment predictions, Logistic Regression also demonstrated superior performance with a score of 0.730, while Naive Bayes achieved 0.677. This shows that Logistic Regression was better at correctly identifying positive sentiment without misclassifying negative or neutral comments as positive.

Similarly, Logistic Regression achieved a higher recall score of 0.722, compared to 0.670 for Naive Bayes. Recall reflects the model’s ability to correctly identify all relevant positive instances, making Logistic Regression more reliable at detecting positive and negative sentiments. When considering F1-score, which combines both precision and recall, Logistic

Regression again showed a better balance with an F1-score of 0.723, surpassing Naive Bayes, which had an F1-score of 0.667. This further supports the conclusion that Logistic Regression is a more robust model for sentiment classification.

However, while Logistic Regression outperformed Naive Bayes in terms of classification performance, it required more time to train. Logistic Regression took 28.3 seconds to complete training, significantly longer than the 4.9 seconds required by Naive Bayes. Despite the increased training time, the superior performance of Logistic Regression across all evaluation metrics justifies the additional computational cost.

The training data distribution was also taken into account, with 41.1% of the comments labeled as negative, 36.7% as neutral, and 22.3% as positive. This distribution ensured that both models were trained on a reasonably balanced dataset, making the performance comparison more meaningful and fair.

In conclusion, based on the evaluation of the models' accuracy, precision, recall, F1-score, and training time, Logistic Regression emerged as the superior model for sentiment analysis. Despite the longer training time, its higher performance across all key metrics made it the optimal choice for deployment. Therefore, Logistic Regression was selected as the final model for sentiment classification, and it was saved along with its corresponding TF-IDF vectorizer and classifier for future use in real-time sentiment analysis applications.

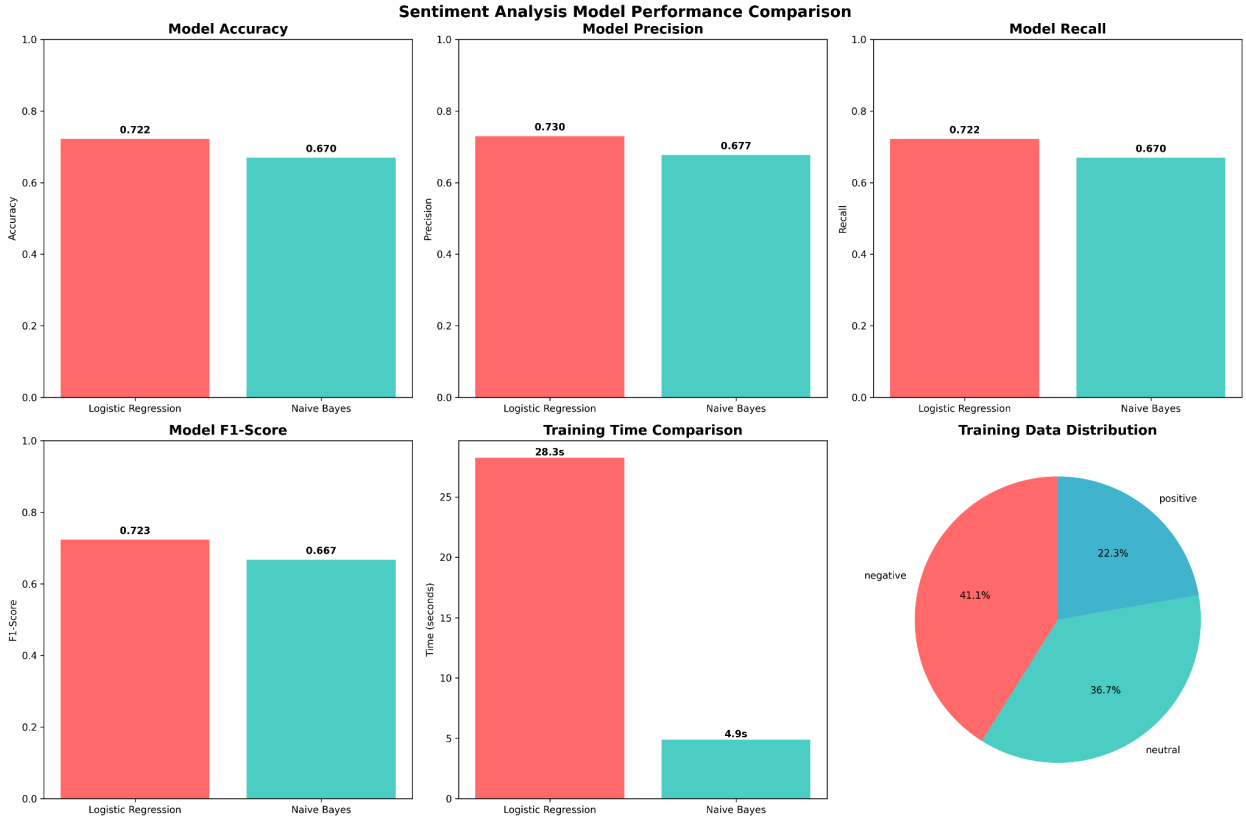

Figure 6.2.2 Sentiment Analysis Model Performance Comparison

7.0 Conclusion & Future Work

This project successfully developed a real-time sentiment analysis system for Malaysian YouTube comments using Apache Kafka and Apache Spark. The system could ingest live comment data, clean the text, and classify sentiments into positive, negative, or neutral using a pre-trained machine learning model. Visualizations in Kibana made it easier to monitor sentiment trends and understand public reactions toward Malaysian-related content. The overall results showed that users generally expressed positive sentiments, especially on historical and geopolitical topics.

In the future, the system can be improved by supporting multiple languages such as Malay and adding better handling for slang or informal language. Advanced sentiment features like emotion classification or sarcasm detection could provide deeper analysis. The model can also be retrained regularly with updated data to improve accuracy. Additionally, expanding data sources beyond YouTube, such as Twitter or Facebook, would help capture a broader view of public opinion in real time.

References

Apache Software Foundation. (2024). Apache Kafka (Version 3.7.0) [Software]. Apache. https://kafka.apache.org/

Apache Software Foundation. (2024). Apache Spark (Version 3.4.2) [Software]. Apache. https://spark.apache.org/

Confluent, Inc. (2024). Confluent Platform Docker Images: cp-zookeeper & cp-kafka (v7.6.1). Retrieved from https://hub.docker.com/r/confluentinc/

Elastic NV. (2024). Kibana (Version 8.13.4) [Software]. Elastic. https://www.elastic.co/kibana/

Elastic NV. (2024). Elasticsearch (Version 8.13.4) [Software]. Elastic. https://www.elastic.co/elasticsearch/

Hugging Face. (n.d.). Transformers: State-of-the-art machine learning for PyTorch, TensorFlow, and JAX. Hugging Face. https://huggingface.co/docs/transformers

Appendix

A. Project Source Code: GitHub Repository

B. Project Configuration: Refer to readme.md

C. Project Logbook: Group 1 GitHub Backlog

D. docker-compose.yml:

```
version: '3.9'

services:
  zookeeper:
    image: confluentinc/cp-zookeeper:7.6.1
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
    ports:
      - "2181:2181"

  kafka:
    image: confluentinc/cp-kafka:7.6.1
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
      - "29092:29092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS:
PLAINTEXT://kafka:29092,PLAINTEXT_HOST://localhost:9092
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
      KAFKA_LISTENERS:
PLAINTEXT://0.0.0.0:29092,PLAINTEXT_HOST://0.0.0.0:9092
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1

  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.13.4
    environment:
      - discovery.type=single-node
```

```
    - xpack.security.enabled=false
    - xpack.monitoring.collection.enabled=true
    - bootstrap.memory_lock=true
    - ES_JAVA_OPTS=-Xms1g -Xmx1g
  ulimits:
    memlock:
      soft: -1
      hard: -1
  ports:
    - "9200:9200"
    - "9300:9300"

kibana:
  image: docker.elastic.co/kibana/kibana:8.13.4
  depends_on:
    - elasticsearch
  ports:
    - "5601:5601"
  environment:
    - ELASTICSEARCH_HOSTS=http://elasticsearch:9200

spark:
  image: bitnami/spark:3.4.2
  depends_on:
    - kafka
    - elasticsearch
  ports:
    - "8080:8080"  # Spark UI
  volumes:
    - ./:/opt/bitnami/spark/work
    - ../data:/opt/bitnami/spark/work/data
  environment:
    SPARK_MODE: master
    SPARK_MASTER_HOST: spark
    SPARK_MASTER_PORT: 7077

ml-trainer:
  build:
    context: .
    dockerfile: Dockerfile.model
```

```
    volumes:
      - ./model:/app/model
      - ../data:/app/data
    environment:
      - PYTHONUNBUFFERED=1
    depends_on:
      - kafka
      - elasticsearch
```

E. Dockerfile.model:

```
FROM python:3.10-slim

# Set working directory
WORKDIR /app

# Install system dependencies
RUN apt-get update && apt-get install -y \
    gcc \
    g++ \
    && rm -rf /var/lib/apt/lists/*

# Copy requirements first for better caching
COPY requirements_model.txt /app/
RUN pip install --no-cache-dir -r requirements_model.txt

# Copy application files
COPY model.py /app/

# Create data and model directories
RUN mkdir -p /app/data /app/model

# Set environment variables for matplotlib (for headless plotting)
ENV MPLBACKEND=Agg

# Run the model training
CMD ["python", "model.py"]
```