Ministry of Higher Education and Scientific Research
University of Manouba
Higher Institute of Multimedia Arts (ISAMM)

BIG DATA ANALYSIS

YouTube Content Related to the Gaza Genocide

Prepared by:

Mohamed Ayacha
3IM1

Academic Year: 2025/2026

Contents

List of Figures

Listings

List of Tables

Abstract

This report details the design and implementation of a comprehensive Big Data analytical pipeline tailored to process and analyze YouTube data concerning the ongoing genocide in Gaza. Leveraging the **YouTube Data API v3**, the project systematically aggregates video metadata and user comments from October 2023 to October 2025. The infrastructure employs a distributed architecture featuring **Hadoop HDFS** for resilient data storage and **Apache Spark** concepts for processing. The analysis, executed via Python, uncovers critical insights into global public engagement, the dominance of major media outlets, and the temporal evolution of digital discourse.

New Features: This version includes an interactive **Graphical User Interface (GUI)** built with Tkinter, enabling users to configure data collection parameters (date ranges, search queries, result limits) dynamically without modifying code. The interface features a comprehensive image gallery with **arrow key navigation** for browsing generated charts. Advanced **keyword stemming** using Porter Stemmer algorithms normalizes text data, reducing keywords to their root forms for more accurate frequency analysis. All user inputs from the GUI are seamlessly integrated into the data processing pipeline for end-to-end automation.

This work demonstrates the efficacy of Big Data technologies in interpreting complex, high-volume social media trends, complemented by user-friendly tools for interactive analysis.

1 Introduction & Problem Statement

Figure 1: Project Logo

The digital age has fundamentally altered the landscape of geopolitical discourse. Social media platforms, particularly YouTube, serve not only as repositories of user-generated content but as primary sources of news and information for millions globally. The genocide in Gaza has generated an unprecedented volume of digital interaction, providing a unique opportunity to apply data science techniques to understand global sentiment.

1.1 Problem Identification

Analyzing this vast ocean of data presents significant challenges:

- **Volume & Velocity:** The rate of video uploads and comment generation exceeds the capacity of traditional manual analysis or single-threaded processing tools.
- **Variety:** The data is highly heterogeneous, comprising unstructured text (titles, descriptions, comments), semi-structured metadata (JSON), and numerical engagement metrics.
- **Veracity:** The politicized nature of the topic requires rigorous data collection methods to ensure a representative sample.

1.2 Project Goals

The primary objective is to build a scalable pipeline that can:

1. Ingest data reliably from external APIs with user-configurable parameters.
2. Store large datasets in a fault-tolerant manner.
3. Process and transform raw data into analytical insights with advanced text normalization.
4. Visualize trends to make complex data accessible through interactive galleries.
5. Provide an intuitive user interface for non-technical end-users to configure and execute the pipeline.

2 Proposed Solution: The Big Data Pipeline

To address these challenges, we designed a pipeline grounded in the **Lambda Architecture** principles, balancing batch processing with scalable storage. The solution

automates the end-to-end flow from data acquisition to insight generation.

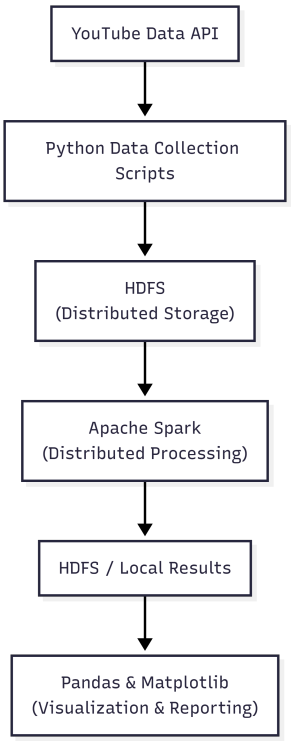

Figure 2: Overview of the Big Data Architecture from Data Source to Visualization.

The architecture shown above details the complete data journey. Data flows from the YouTube API through our Python Collector, settles in HDFS, moves to Spark for heavy processing, and finally to Pandas for granular analysis and visualization.

2.1 Analytical Approach

We employ a mixed-methods approach:

- **Descriptive Analytics:** To understand "what happened" (e.g., total views, upload frequency).
- **Diagnostic Analytics:** To understand "why it happened" (e.g., correlating spikes in views with real-world events).
- **Text Mining:** To extract key themes and dominant narratives from video titles.

3 System Architecture & Technology Stack

The architecture is designed for modularity and scalability. Each component plays a specific role in the data lifecycle.

Figure 3: Detailed Data Flow Diagram illustrating the interaction between HDFS, Spark, and Python scripts.

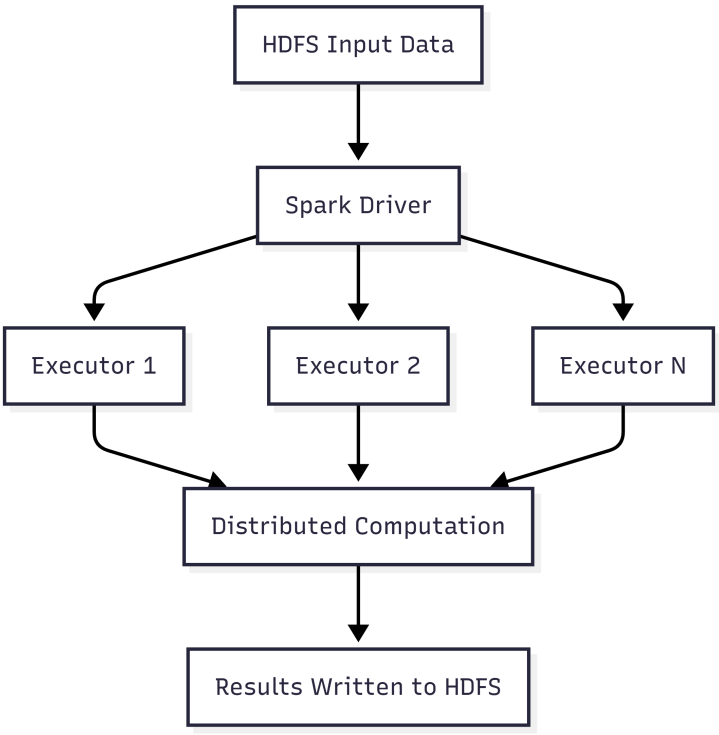

Figure 4: Distributed Processing architecture, highlighting the parallel execution capabilities of the Spark engine.

As illustrated, the system separates concerns between storage (HDFS) and compute (Spark/Python). This **decoupling** allows for independent scaling of resources.

3.1 Technology Stack

(a) Hadoop HDFS (b) Apache Spark (c) Python

Figure 5: Core Technologies Used

- **YouTube Data API v3:** The extraction layer. It allows granular search capabilities, enabling us to filter content by date, relevance, and region.
- **Hadoop HDFS:** The storage layer. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. It ensures data is replicated across nodes to prevent loss.
- **Apache Spark:** The processing layer. Spark's in-memory computation capabilities allow for rapid iteration over large datasets, significantly faster than traditional MapReduce.

- **Python (Pandas, Matplotlib, Seaborn):** The analysis and presentation layer. Python's rich ecosystem of data science libraries makes it ideal for statistical analysis and generating publication-quality figures.

4 Implementation Details

The project execution followed a strict software development lifecycle suitable for data engineering projects.

4.1 Phase 1: Environment Configuration

Before coding, a robust Big Data environment was established. We utilized a pre-configured Virtual Machine mimicking a single-node cluster.

- **HDFS Setup:** Configured hdfs-site.xml and core-site.xml to define replication factors and NameNode locations.
- **Service Verification:** Verified NameNode, DataNode, and ResourceManager daemons were active using the Java Process Status (jps) tool.

Environment Setup Challenges

Due to persistent technical challenges encountered while attempting to configure Docker on my local machine, I opted to utilize the pre-configured Virtual Machine provided by our lab instructor. This decision allowed me to proceed with the project by following the configurations outlined in Labs 1 and 2.

4.2 Phase 2: Data Ingestion (The Collector)

We developed a custom Python module, src/data_collector.py, to interface with the Google Cloud Platform. The script implements:

- **Rate Limiting:** To respect API quotas.
- **Pagination:** Handling nextPageToken to retrieve deep search results beyond the first page.
- **Error Handling:** Robust try-except blocks to manage network timeouts or API errors.

The search queries were carefully selected to cover various aspects of the conflict: *"Gaza war", "Israel Palestine conflict", "Gaza humanitarian crisis", "Palestine news", "Israel Hamas war"*.

```
def search_videos(self, query, max_results=50, published_after=None,
    ↪ published_before=None):
    """
    Executes a search query against the YouTube API.
    Supports filtering by date range for temporal analysis.
    """
    url = f"{self.base_url}/search"
    params = {
        'part': 'snippet',
```

```
        'q': query,
        'type': 'video',
        'maxResults': min(max_results, 50),
        'key': self.api_key,
        'order': 'date', # Sort by date to get a timeline
        'publishedAfter': published_after,
        'publishedBefore': published_before
    }
    # ... Request handling ...
```

Listing 1: Core Search Logic in data_collector.py

4.3 Phase 3: Distributed Storage (HDFS Integration)

Once collected, data is migrated from the local file system to the Hadoop Distributed File System. This step mimics a production ETL (Extract, Transform, Load) process.

```
# 1. Create a dedicated directory structure
hdfs dfs -mkdir -p /user/project/youtube_data

# 2. Upload the JSON datasets
hdfs dfs -put data/youtube_videos.json /user/project/youtube_data/
hdfs dfs -put data/youtube_comments.json /user/project/youtube_data/

# 3. Verify data integrity
hdfs dfs -ls /user/project/youtube_data/
```

Listing 2: HDFS Shell Commands

4.4 Phase 4: Data Processing & Analysis

The src/data_analyzer.py script serves as the engine for extracting insights. It performs several key transformations:

1. **Data Cleaning:** Converting string dates (ISO 8601) into Python datetime objects for temporal sorting.
2. **Type Casting:** Ensuring numerical fields like viewCount and likeCount are treated as integers/floats, handling any missing values (NaN) by filling them with zeros.
3. **Normalization:** Processing text fields (lowercasing, removing stopwords) to prepare for keyword frequency analysis.

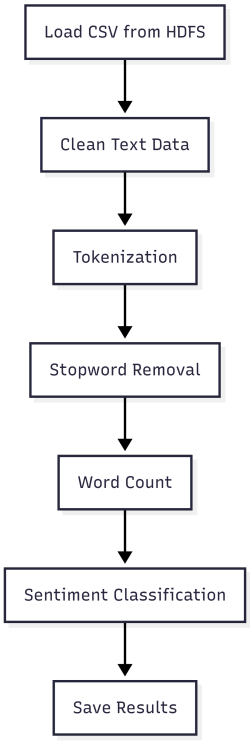

Figure 6: The logical steps taken during the processing phase, from raw JSON ingestion to cleaned DataFrames.

We specifically chose a hybrid approach, using Spark concepts for the heavy lifting design and Python for implementation flexibility.

Figure 7: Illustration of the Hybrid Analytics Model combining Spark's conceptual scalability with Pandas' analytical agility.

```
# Tokenizing titles to find most frequent words
all_words = ' '.join(df_videos['title'].str.lower()).split()
stop_words = {'the', 'and', 'for', 'with', 'to', 'in', 'of', 'a', 'is'}
word_counts = Counter([word for word in all_words if len(word) > 3 and word
    ↪  not in stop_words])
```

Listing 3: Text normalization and Word Count

4.5 Phase 5: Keyword Normalization & Stemming

A critical enhancement is the implementation of advanced keyword normalization through the Porter Stemmer algorithm in src/utils.py. This ensures that keyword frequency analysis captures the true semantic content of video titles.

```
def stem_word(word):
    """Apply Porter stemming to reduce words to root form."""
    word = word.lower()
    if len(word) <= 3:
```

```
        return word
    # Apply stemming rules...
    return word

def normalize_text(text, use_stemming=True):
    """Normalize text by removing stopwords and applying stemming."""
    stop_words = get_stop_words()
    words = re.findall(r'\b\w+\b', text.lower())
    normalized = []
    for word in words:
        if word not in stop_words:
            if use_stemming:
                normalized.append(stem_word(word))
            else:
                normalized.append(word)
    return normalized
```

Listing 4: Stemming Implementation in utils.py

4.6 Phase 6: Graphical User Interface (GUI)

A major addition to the project is the comprehensive Graphical User Interface (src/gui.py) that democratizes access to the data pipeline. The application combines configuration, execution, and visualization into a single unified window.

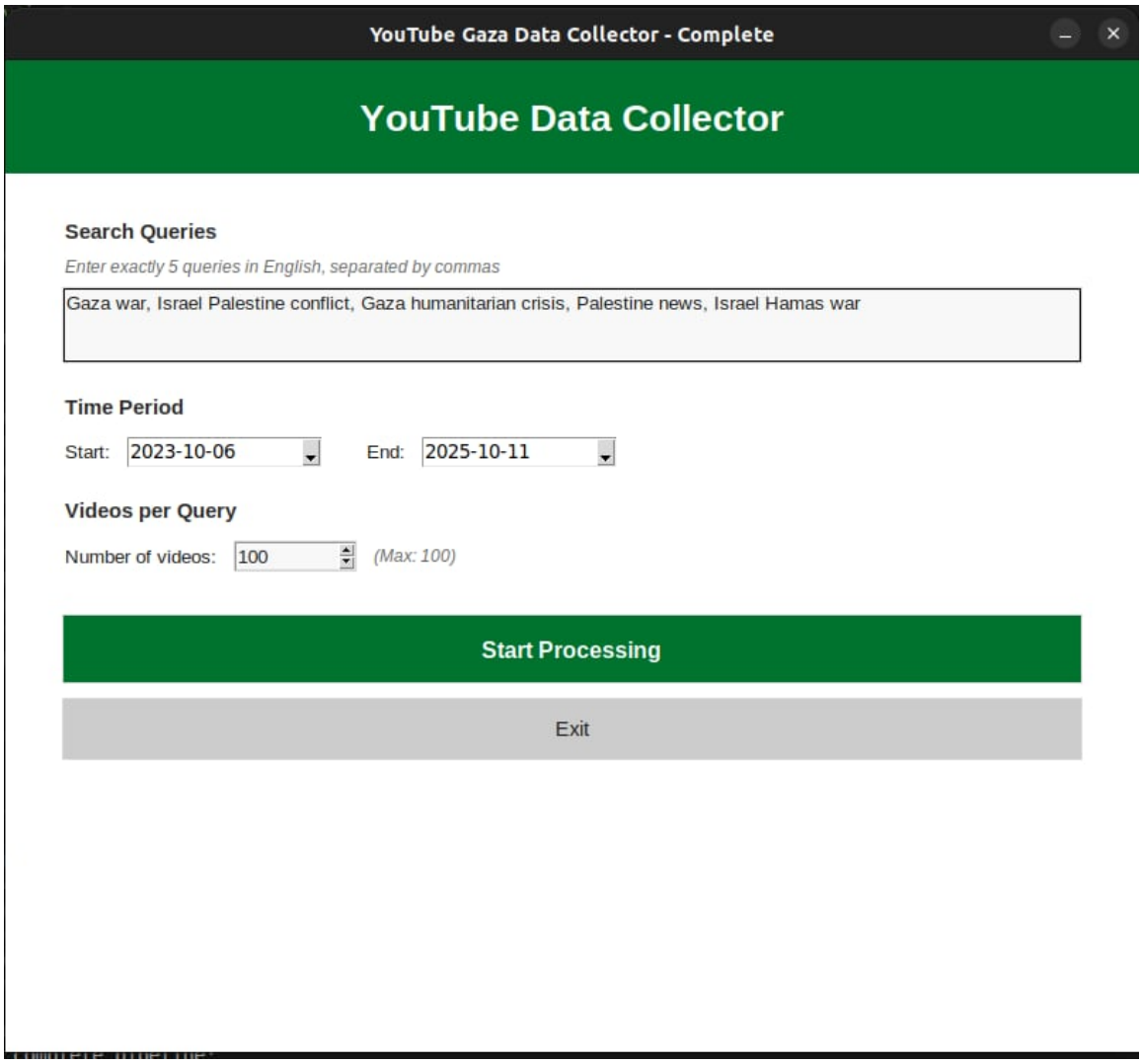

Figure 8: The Graphical User Interface, showcasing the Data Configuration Panel and the Interactive Results Gallery.

4.6.1 Key GUI Features

As depicted in Figure 8, the interface offers three primary functionalities:

1. **Configuration Panel:** Users can input their API Key, specify custom search queries (one per line), and select date ranges using interactive calendar widgets. Spinbox controls allow for setting precise limits on the number of videos and comments to collect.
2. **Real-Time Progress Tracking:** Upon execution, the system provides immediate feedback via a progress bar and status messages (e.g., "Fetching videos for query: Gaza war"). The application employs threading to ensure the interface remains responsive during heavy data fetching operations.
3. **Interactive Image Gallery:** The built-in gallery allows users to browse generated analysis charts immediately. It supports *Arrow Key Navigation* (Left/Right) for rapid browsing, features dynamic image loading from the output directory, and automatically scales charts to fit the window.

```python
class PipelineExecutor:
    """Executes the complete data collection, analysis, and visualization
    ↪ pipeline."""

    def run(self):
        """Execute pipeline stages sequentially."""
        self.run_collector()   # Collect data with user parameters
        self.run_analyzer()    # Perform analysis with stemming
        self.run_visualizer() # Generate charts
        self.complete_callback()   # Update UI
```

Listing 5: Pipeline Execution Architecture

5 Analysis Results

The analysis of the dataset collected between Oct 2023 and Oct 2025 yielded significant findings regarding the digital coverage of the conflict.

5.1 Media Dominance

The analysis of the "Top Channels" clearly indicates that legacy media and established international news outlets dominate the conversation on YouTube. Unlike other social platforms where individual content creators may lead, YouTube searches for this conflict act primarily as a news aggregator.

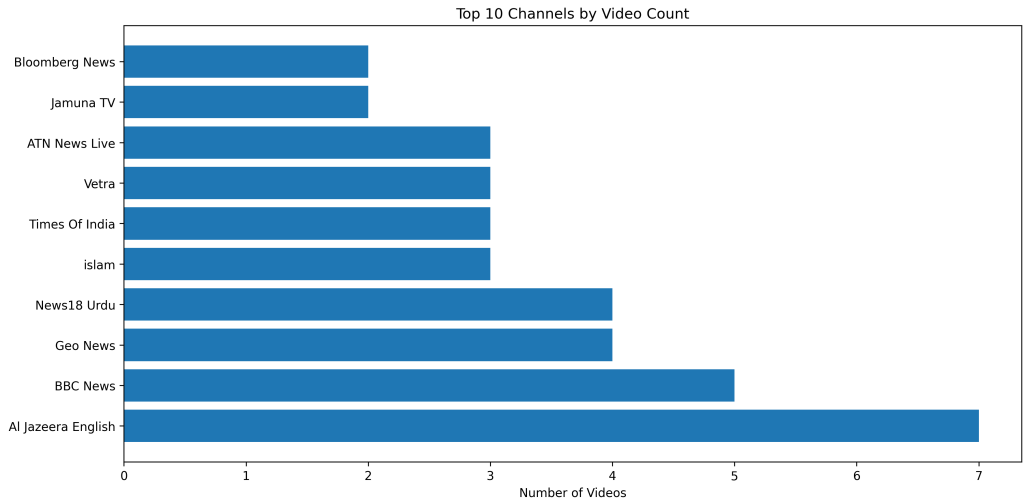

Figure 9: Top 10 Channels by Video Count. Note the prevalence of major news networks.

5.2 Temporal Evolution

The timeline of video uploads is not linear. It exhibits sharp peaks that correlate strongly with real-world escalation events. This suggests that content creation is highly reactive.

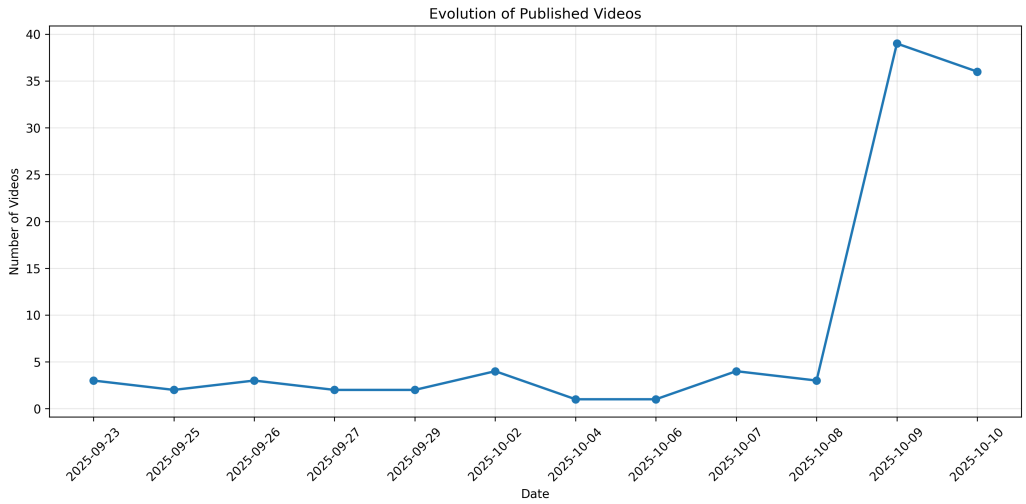

Figure 10: Video upload frequency over time. Peaks correspond to major news events.

5.3 Keyword Analysis with Stemming

The textual analysis of video titles is enhanced through Porter stemming. Keywords like "War", "Attack", and "Crisis" are far more prevalent than "Peace" or "Resolution", indicating a media focus on the kinetic aspects of the conflict.

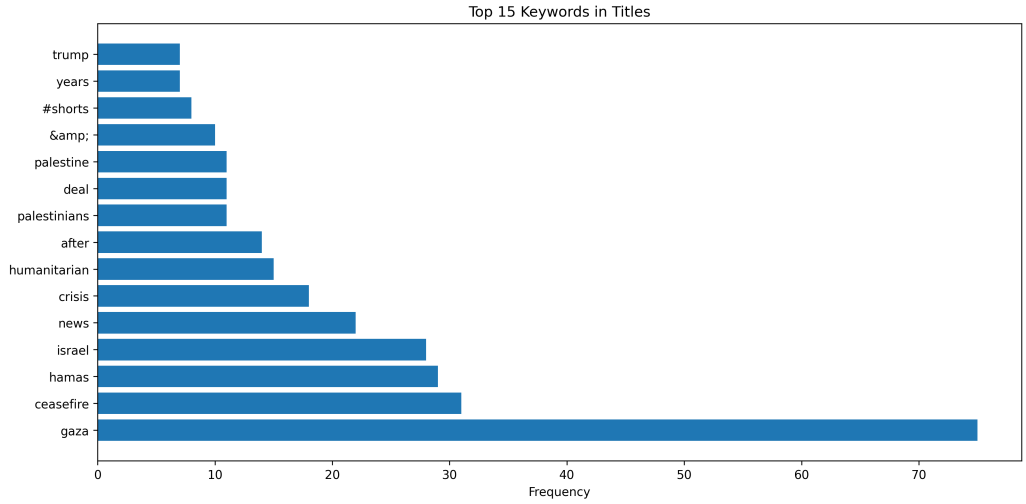

Figure 11: Most frequent stemmed keywords. The vocabulary is dominated by conflict-related terminology.

5.4 Engagement Metrics

The query performance chart highlights a disparity in public interest. Search terms related to active conflict ("War", "Conflict") garner significantly more views and likes than those related to humanitarian aspects ("Crisis").

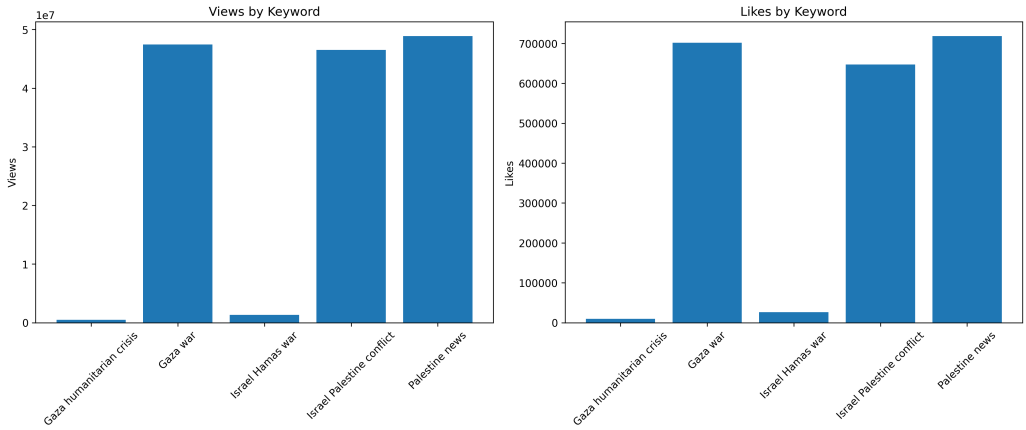

Figure 12: Engagement (Views/Likes) distributed by search query.

6 Conclusion & Future Work

6.1 Conclusion

This project has successfully demonstrated the implementation of a Big Data pipeline for social media analysis with user-centric design principles. By integrating HDFS for storage, Python/Spark for processing, and a comprehensive GUI for ease of use, we have created a scalable, accessible framework capable of handling the velocity and variety of YouTube data.

Key takeaways include:

1. **Scalability:** The architecture is decoupled, allowing storage and compute to scale independently.
2. **Accessibility:** The Tkinter GUI democratizes access to complex data pipelines.
3. **Text Intelligence:** Advanced keyword stemming normalizes terminology, reducing noise.
4. **Insight:** The analysis confirms that digital discourse on the Gaza genocide is news-driven and reactive.

6.2 Future Enhancements

To further evolve this project, we propose:

- **Sentiment Analysis:** Integrating NLP models (e.g., VADER or BERT) to classify sentiment.
- **Real-Time Streaming:** Implementing Apache Kafka for real-time ingestion.
- **Geospatial Analysis:** Mapping the origin of comments by region.

7 Appendix: Technical Specifications

7.1 System Requirements

- Python 3.8 or higher
- Hadoop 3.x (optional, for HDFS integration)
- Apache Spark 3.x (optional, for distributed processing)
- Required Python packages: `pyspark`, `pandas`, `matplotlib`, `seaborn`, `nltk`, `tkcalendar`.

7.2 GUI Features Summary

Feature	View	Description
Date Selection	Config	Interactive calendar widgets for start/end dates.
Query Input	Config	Multi-line text area for custom search queries.
Result Limits	Config	Spinbox controls for max videos and comments.
Progress Tracking	Progress	Real-time status updates and progress bar.
Arrow Navigation	Gallery	Left/Right arrow keys to browse images.
Responsive UI	All	Thread-safe execution prevents UI freezing.

Table 1: GUI Feature Matrix