Hindustan Institute of Technology and Science

Code Crunch 2024

(National Level Hackathon)

Low-Level Design

on

Sentiment Analysis with Multi-Lingual Support

Team Name : Team Sharingan

Team members:
Ashwath Krishna U - 21137025
Pranav Kumar P - 21137034
Tejo Sharan A M - 21137048
Hirthik G - 21137052
Yoga Darsa V M - 21137073

Table of Contents

CHAPTER 01 - INTRODUCTION

1.1 Scope of the Document

This document outlines the structure, implementation, and functional behavior of a Real-Time Multi-Language Sentiment Analysis Tool using Twitter data. The tool grasps the Twitter API, natural language processing (NLP) techniques, and pre-trained sentiment analysis models to track and analyze the sentiment of tweets in real time. The application is accessible via a web interface built using Streamlit, allowing users to input specific keywords, retrieve tweets, and view their sentiment analysis.

1.2 Intended Audience

This document is intended for the following groups:

- **Developers**: Engineers who are responsible for maintaining and extending the sentiment analysis tool. This includes handling API integration, NLP models, and web UI components.
- **Data Scientists**: Analysts who will fine-tune the sentiment model and ensure its effectiveness across multiple languages.
- **Product Managers**: Individuals responsible for the strategic development of this tool, ensuring it meets the business needs for monitoring and analyzing social sentiment.
- **End Users**: Individuals and organizations interested in tracking and analyzing real-time sentiment trends on Twitter, such as marketing professionals, customer service teams, or social media strategists.

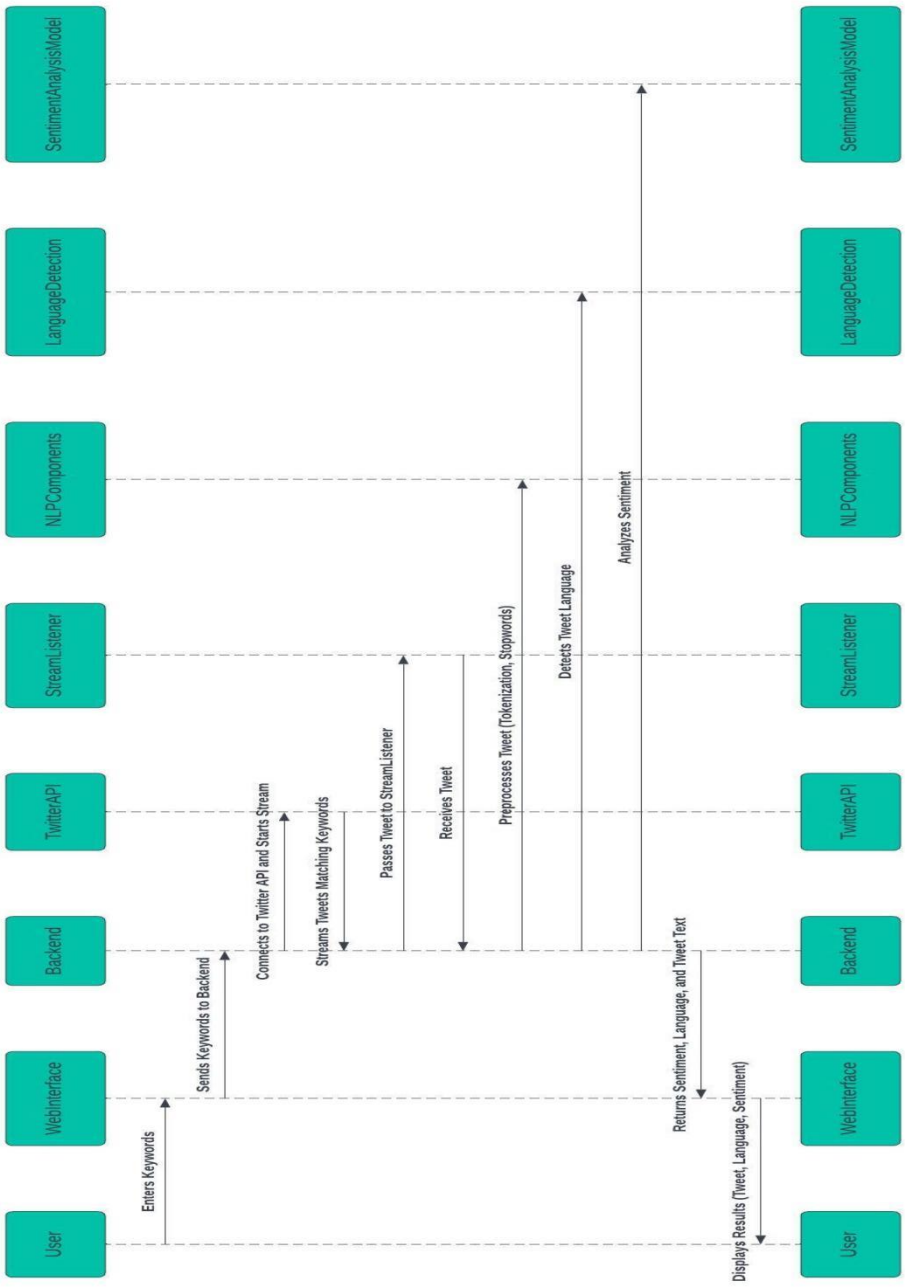
1.3 System Overview

The **Real-Time Sentiment Analysis Tool** offers the following features:

- **Twitter Data Integration**: The system uses the Twitter API to stream real-time tweets based on user-specified keywords or hashtags.
- **Language Detection**: The tool automatically detects the language of each tweet using the langdetect library, ensuring appropriate processing for multilingual data.
- **Preprocessing**: The system preprocesses each tweet by removing stopwords and performing tokenization using the nltk library. This step ensures that only meaningful parts of the text are passed to the sentiment analysis model.
- **Sentiment Analysis**: The tool leverages the Hugging Face transformers pipeline to perform sentiment analysis. The model classifies the sentiment of each tweet as Positive, Negative, or Neutral.
- **Web Interface**: The tool is built with Streamlit, providing a simple and interactive web interface where users can input keywords, view live-streaming tweets, and monitor sentiment results in real-time.

CHAPTER 02 - LOW LEVEL SYSTEM DESIGNS

2.1 Sequence Diagram



2.2 Navigation flow / UI Implementation

Navigation Flow

The user interface (UI) for the Real-Time Sentiment Analysis Tool is built using Streamlit, which offers an intuitive flow with minimal navigation complexity. The navigation follows a simple sequence of interactions:

1. **Homepage (Real-Time Sentiment Analysis)**:

- Title and brief description of the tool.
- An input field where the user can enter keywords or hashtags to track.
- A "Start Stream" button to begin the real-time tweet analysis based on the entered keywords.

2. **Real-Time Results Screen**:

- Once the user starts the stream, the results section below the input field updates in real-time with:

    1. Tweet text.
    2. Language detected.
    3. Sentiment analysis result (Positive, Negative, Neutral).
    4. Sentiment confidence score.

- An optional button (e.g., "Stop Stream") to stop the current analysis.

UI Implementation

- **Streamlit** automatically handles routing, so a single-page interface dynamically updates with user inputs.
- Components such as text boxes, buttons, and display sections are implemented in the Streamlit app.
- Real-time updates are handled by re-rendering the output as new data (tweets and sentiments) stream in.

2.3 Screen Validation, defaults, and Attributes

Screen Validations:

- **Keywords Input**:

    1. Must not be empty.
    2. Minimum number of characters required: 1.
    3. Special characters are allowed (since hashtags, keywords, or usernames can contain them).

Defaults:

- The keyword input field has placeholder text: *"Enter keywords to track (comma-separated)"*.
- The initial state of the results section is hidden and only becomes visible once the "Start Stream" button is pressed.

Attributes:

- **Input Field (Text Input)**: A single text field where users can input one or more keywords separated by commas.
- **Button (Start Stream)**: Initiates the real-time tweet analysis.
- **Results Display Area**: A dynamically updated section that displays the incoming tweets and sentiment analysis results.

2.4 Client Side Validation Implementation

Client-side validation is implemented using Streamlit's form capabilities and Python-based checks. The following validations will occur before data is submitted to the backend:

- **Keyword Input Validation**:
  - If the input is empty or contains only whitespace, display an error message: “Please enter at least one keyword.”
  - If keywords contain only special characters or non-alphanumeric data, the user will be prompted to enter valid text.

- **Real-time Error Feedback**: If the user enters invalid data, the error will be shown instantly without requiring a page refresh, ensuring a smooth user experience.

2.5 Server - Side Validation Implementation

Server-side validation occurs once the client submits data. These validations are crucial for ensuring security, API limits, and consistent processing of input. The backend Python code will handle:

Rate Limiting Validation:

- Ensure that the tool does not exceed the Twitter API's rate limits.
- If limits are exceeded, an appropriate message will be shown, like “Twitter API rate limit exceeded, please try again later.”

Keyword Format Validation:

- Ensure that the input is not only non-empty but also correctly formatted (e.g., contains valid characters).
- Convert the keywords to a format compatible with the Twitter API (e.g., handling commas between keywords).

Tweet Language Validation:

- If the tweet language is not supported, the backend will log it and skip sentiment analysis for that tweet.

2.6 Components Design Implementation

Frontend Components (Streamlit):

Keyword Input Field:

1. A text input field (st.text_input) where users enter keywords.
2. Example: keywords_input = st.text_input("Enter keywords to track", "Python, AI").

Start Stream Button:

1. A button (st.button) that starts the real-time stream.
2. Example: if st.button("Start Stream").

Results Display Section:

1. A dynamic section (st.write) that displays each tweet, its detected language, and sentiment.

Backend Components (Python & Tweepy):

Twitter API Connection:

1. Tweepy handles the connection to the Twitter API and listens for tweets based on the keywords provided.

Streaming Listener:

1. A custom stream listener class that listens to real-time tweets and processes them for sentiment analysis.

Language Detection:

1. Utilizes the langdetect library to automatically detect the language of each tweet.

Sentiment Analysis Pipeline:

1. A pre-trained sentiment analysis model using the Hugging Face pipeline for real-time sentiment classification.

2.7 Configurations / Settings

Twitter API Settings:

- **API Keys**: Store API credentials (e.g., consumer_key, consumer_secret, access_token, access_token_secret, and bearer_token) in environment variables for security and ease of configuration.

Model Configurations:

- **Sentiment Model**: Configure the sentiment model to use either a multilingual transformer model (e.g., mBERT, XLM-RoBERTa) or language-specific models depending on user needs.

Stream Settings:

- Set keyword filtering rules for the streaming client to limit the tweets processed (based on user input).

NLTK Settings:

- Download and store necessary datasets for stopword removal and tokenization in the local environment to ensure preprocessing works correctly (nltk.download("punkt"), nltk.download("stopwords")).

2.8 Interfaces to other components

- **Twitter API**:

  Interface between the **Backend (Tweepy)** and **Twitter** to stream tweets in real-time. Tweepy connects to the Twitter API using OAuth authentication and streams tweets matching the user's keyword input.

- **Language Detection**:

  Interface between the **Backend (Python Application)** and the **Langdetect Library** for detecting the language of each tweet.

- **Sentiment Analysis Model**:

  Interface between the **Backend** and the **Transformers library** for processing preprocessed tweets and classifying sentiment in real-time.

- **Streamlit Frontend**:

  Interface between the **Backend** and **Frontend (Streamlit)**, which displays the processed results to the user in real-time.

CHAPTER 03 - DETAILS OF OTHER FRAMWORKS BEING USED.

3.1 Session Management

Overview

Session management is responsible for maintaining the state of a user's interactions across multiple requests in the web application. In the case of the real-time sentiment analysis tool, session management ensures that user inputs (e.g., keywords, preferences, and results) are retained across interactions without requiring the user to re-enter information.

Implementation Approach

1. **Stateless Framework**: **Streamlit** is a stateless framework by default, meaning it does not automatically preserve the state across user actions. Therefore, custom session management is required to maintain the user's inputs and analysis history.
2. **Session State Management in Streamlit**: Streamlit provides a SessionState feature that allows you to store variables such as the current user, search keywords, and streaming status across multiple requests.

Steps for Session Management in Streamlit:

- **Store user inputs**: For each session, the keywords the user enters should be stored in the session state. This allows the app to retain keywords even if the page refreshes or multiple actions are taken.
- **Store streaming status**: The state of whether the stream is running or stopped can be maintained in the session state to provide continuity.
- **Store analysis results**: Analysis results should be stored in the session until the user ends the session or logs out.

Session Expiry:

- Implement session timeouts to automatically end the session if the user is inactive for a specific duration. This can be done by checking the last activity timestamp and clearing the session if the user is inactive for a predefined period (e.g., 30 minutes).

Security Considerations:

- **Session hijacking prevention**: Secure sessions with HTTPS to prevent session hijacking.
- **Session tokens**: Generate unique session tokens for each user and store them securely (if user authentication is implemented).
- **Session expiry**: Automatically expire sessions after a set period to prevent prolonged access.

3.2 Caching

Overview

Caching is essential to improve performance, especially when dealing with real-time data and frequent API calls (e.g., to the Twitter API). By caching frequently accessed data, the application can reduce the load on external services and speed up response times for users.

Types of Data to Cache:

Twitter API Data:

1. Cache recently fetched tweets to avoid making duplicate API requests for the same keywords within a short period.

Sentiment Analysis Results:

1. Cache sentiment analysis results for each tweet, so repeated analysis of the same tweet doesn't require recomputation.

User-Specific Data:

1. Cache user-specific data, such as previous search keywords and recent sentiment results, to enhance the user experience when they revisit the application.

Implementation Approach:

In-Memory Caching (Redis or Streamlit Cache):

1. Use an in-memory caching system like **Redis** to store frequently accessed data. Alternatively, Streamlit's st.cache_data decorator can be used to cache results within the app.

Database Caching:

1. Store frequently accessed data (e.g., past analysis results) in a database to minimize the need for redundant API calls or sentiment analysis.

Cache Expiry:

1. Set appropriate cache time-to-live (TTL) values based on the frequency of data changes. For example:
    1. **Twitter data**: Since tweets are frequently updated, cache for a short period (e.g., 5-10 minutes).
    2. **Sentiment analysis**: Cache results for a longer period (e.g., 24 hours), as they are less likely to change for a given tweet.

Invalidation Strategy:

1. Implement a cache invalidation mechanism to clear stale data. For example, if a user searches for new keywords, invalidate the old cached results.

Security Considerations:

- **Sensitive Data**: Ensure that sensitive information, such as API credentials or personal user data, is not cached.
- **Cache Poisoning**: Implement security checks to prevent cache poisoning attacks, where malicious data might be inserted into the cache.

CHAPTER 04 - UNIT TESTING

For the real-time sentiment analysis tool using the Twitter API, unit testing ensures that individual components such as the tweet fetching, sentiment analysis, language detection, and pre-processing logic are functioning correctly. Unit tests can also be designed to verify session management, caching, and any error handling mechanisms.

Unit Testing Framework

You can use unittest, Python's built-in testing framework, along with mock to simulate external services like the Twitter API and language detection library. Alternatively, you can use pytest for more advanced features and easier test writing.

CHAPTER 05 - KEY NOTES

Twitter API Limits: Remember that Twitter has rate limits on API usage. Ensure you do not exceed those limits during testing.

Sentiment Analysis: The pre-trained model from Hugging Face has some limitations based on the data it's trained on. For extensive multilingual analysis, consider using specialized models if available.

Real-Time Stream Handling: The example above runs the stream asynchronously. Depending on your environment and the amount of data, you may want robust error handling and connection management.

Environment Setup: Running this demo requires the appropriate environment with API access; ensure your credentials are correctly set in the script.

CHAPTER 06 - REFERENCES

1. A Comprehensive Review on Sentiment Analysis

https://arxiv.org/abs/2311.11250

2. Survey of Sentiment Analysis

https://www.mdpi.com/2076-3417/13/7/4550

3. Survey on Sentiment Analysis Method, Applications and Challenges

https://link.springer.com/article/10.1007/s10462-022-10172-5

4. Multi-Lingual Text Recognition system.

https://link.springer.com/article/10.1007/978-981-99-5221-2_36

5. Lingualyzer

https://link.springer.com/article/10.3758/s13428-023-02096-9