**Current Setup:**

The existing sentiment analysis application processes plain text entered by user in Text Box in UI or txt files uploaded by user:

1. **UI**: We have utilized plain HTML and JavaScript to fetch input from user as text box in put or txt file uploads. The output from model, category is received as Json data and scores are received as a graph image. Both values are processed and displayed using html and JavaScript.
2. **Backend & API**
   1. **API:** We are using Flask framework to implement a python API for the business logic layer.
   2. **Python libraries used:** NLTK for sentiment analysis, flask for API and matplotlib for visualization
   3. **Preprocessing the text**: We have implemented following pre-processing of the text before supplying the text for sentiment analysis.
3. Tokenization
4. lemmatization, and
5. stop words removal
   1. **Sentiment analysis**: We have used NLTK library for sentiment analysis. The model calculates compound scores based on the vader\_lexicon
      * The compound score is a single number between -1 and 1.  
        Score closer to 1 indicates a positive sentiment.  
        Score closer to -1 indicates a negative sentiment.  
        Score near 0 indicates a neutral sentiment.

**Enhancing to Real-Time Sentiment Analysis**

**Real-Time Input Source:**

For real-time sentiment analysis, we need a continuous stream of text data. As a very widely used example we can use APIs provided by social media platforms like: Twitter's API for real-time tweets OR streaming messages from messaging platforms like Slack. We can also filter certain category of live tweets based on specific keywords in the Twitter Streaming API.

* **Available libraries:** tweepy, an open-source Python package using which you can access the Twitter API.
* **Specific class implementation:** Creating a class inheriting from **StreamListener** class of **tweepy** library

**User Interface (UI)**

**More suitable frameworks:**

We can go for popular frameworks like ReactJS or Angular for UI implementation.

**Integrated WebSocket:**

This will improve the performance and responsiveness which is an advantage for real time applications.

* **Available libraries: fastapi**, **WebSocket** can be implemented which will allow for real-time communication between a client and server.

**Asynchronous Processing:**

As we are handling live streaming data, we can make the sentiment analysis asynchronous process so that the queries don’t wait for other queries to complete.

1. **Available libraries: asyncio**, framework can be utilized to make the sentiment analysis queries asynchronous.

**Modify Preprocessing for Streaming Data using Regex:**

**Real-time Cleaning**: We can use regular expressions to process, slang, emojis, or special characters in real time before supplying the content to the model.

**Partial Text Handling**: We can also do partial text handling to manage incomplete sentences differently.

**Retaining other preprocessing steps:**

Preprocessing steps we have implemented now, Tokenization, lemmatization and stop words removal are still relevant and can be retained during live processing as well.

**Message queues:**

Instead of sending the text/content directly through API calls we can implement message queues like Kafka or RabbitMQ to distribute processing and have stable queueing system.

On traffic handling side, Stream Buffering and load balancing can be implemented for a more stable application to manage sentiment analysis for live streaming data.