Comment Processing Document & API Setup Note

# Background

**Code**

All the code and datasets are accessible through the GitHub repo:

<https://github.com/CYS721/myLivingCity>

**Comment Funnel Function:** The primary deliverable will be the development and implementation of the Comment Funnel Function within the MyLivingCity platform. This function will automatically suggest similar existing comments to users during the comment submission process, thereby reducing duplicate content and enhancing the clarity of discussions. The system will prompt users to agree with existing comments rather than posting duplicates, streamlining the conversation flow and emphasizing community consensus.

**Input Filtering System:** Another critical deliverable is the creation of a robust input filtering system designed to detect and prevent the submission of inappropriate comments. This system will use advanced text analysis techniques to identify offensive language, off-topic remarks, and other unwanted content. A warning message will prompt users to modify their inputs accordingly, ensuring that the dialogue remains constructive and relevant to the community's interests.

We developed the comment funnel function based on a “YouTube Comments” dataset.

YouTube comments include many elements, including but not limited to internet slang, emoticons, simplified language, emoji expressions, uncivil language (profanity), and lengths ranging from a single word to an entire paragraph. In summary, YouTube comments are an excellent model training corpus.

We developed the comment filter function based on a publicly accessible package “profanity-check”.

# Function description

## Funnel function

**Input**:

A string (the user's comment)

**Output**:

A list of 7 keywords in the following format:

* Tone (Positive, Negative, or Neutral)
* Attitude (verb)
* Keyword1
* Keyword2
* Keyword3
* Keyword4
* Keyword5

A similar comment from existed comment:

* Comment (or None)

**Example Input**:

I love the product's quality and customer support!

**Example Output**:

0. Positive | 1. appreciates | 2. product | 3. quality | 4. customer | 5. service | 6. support

The quality of this product is amazing and I received great support from the service.

## Filter function

**Input**:

A string (the user's comment)

**Output**:

Profane / Not Profane

**Example Input**:

I love the product's quality and customer support!

**Example Output**:

Not Profane

# API Requirement

## Funnel function

**API Request**

POST /funnel\_function HTTP/1.1

Content-Type: application/json

{

"comment": "I love the product's quality and customer support!",

“idea\_id”: 133

}

**API Response**

HTTP/1.1 200 OK

Content-Type: application/json

{

"tone": "Positive",

"attitude": "appreciates",

"keywords": [

"product",

"quality",

"customer",

"service",

"support"

]

"similar\_comment": "The quality of this product is amazing and I received great support from the service."

}

## 

## Filter function

**API Request**

POST /filter\_function HTTP/1.1

Content-Type: application/json

{

"comment": "I love the product's quality and customer support!"

}

**API Response**

HTTP/1.1 200 OK

Content-Type: application/json

{

"check": "Not Profane"

}

# My understanding of the comment processing

Initially, there is some data on the website, and developers can use our funnel function to process the existing comments one by one. Our function will return the corresponding keywords for each comment, which developers can store in a database (using a separate table). This way, when new comments are added to the website later, they can be compared and analyzed.

After this, every time a new comment is submitted to the website, developers can first use the filter function to check if the comment contains any non-compliant language, and then use the funnel function to find similar existing comments that convey similar meanings.

Therefore, this also raises another issue that needs to be discussed: We need website developers to provide a way to access the original comments. This way, we can compare new comments with the existing ones and find similar comments.

To facilitate efficient deployment and simplify the development process, we can containerize the code by creating a Dockerfile. This enables developers to easily pull the image and deploy it in a production-ready environment on a server.