

# SafeFeed

Context-Aware Content Moderation  
with Generative AI

## Team Members

Sruthi Bhaskar

Nishanth Prasath

Lakshman Raaj Senthil Nathan

# Table of Contents

	<b>Introduction</b>
	<b>Ideation</b>
	<b>Scraping Reddit Policies for chatbot and submission integrity checks</b> 1. Function: scrape_reddit_policies
	<b>Processing (Trials)</b> 1. Function: vectorize_policies 2. Function: similarity_search 3. Function: check_policies
	<b>Processing (Actual)</b> 1. Policy Checker Assistant 2. OpenAI Assistant Overview 3. Detailed Function Descriptions 3.1. Function: list_files 3.2. Function: create_thread 3.3. Function: send_message 3.4. Function: wait_on_run 3.5. Function: analyze_content_image 3.6. Function: analyze_content_text 3.7. Function: get_responses 3.8. Function: extract_information_image 3.9. Function: extract_information_text 4. Detailed Workflow 4.1. reddit_scrapper_v2 - Data Loader 4.2. submission_content_moderation - Custom Transformer 4.3. reddit_comments_scraper - Data Loader 4.4. combine_categories - Transformer 4.5. comment_content_moderation - Transformer 4.6. store_data - Data Exporter 4.7. combine_categories_comment - Transformer 4.8. store_comments_data - Data Exporter
	<b>Policy guide Chatbot</b> 1. Policy guide assistant 2. OpenAI Assistant Overview 3. Detailed Function Descriptions 3.1. Function: load_openai_client_and_assistant 3.2. Function: wait_on_run 3.3. Function: get_assistant_response 3.4. Function: add_message 4. Detailed Workflow

	<b>Policies Customizer</b> <ol style="list-style-type: none"> <li>Detailed Function Descriptions <ol style="list-style-type: none"> <li>Function: openai_connection</li> <li>Function: initialize_default_policies_and_folders</li> <li>Function: load_policies</li> <li>Function: save_policies</li> <li>Function: get_file_id</li> <li>Function: update_file_openai</li> </ol> </li> <li>Detailed Workflow</li> </ol>
	<b>LLM Selection process</b> <ol style="list-style-type: none"> <li>Claude Response</li> <li>Jurassic2 from AI21Studio labs response</li> <li>OpenAI Assistants (GPT4) response</li> </ol>

## Introduction

In the ever-evolving landscape of digital communities, the task of moderating user-generated content has emerged as a paramount challenge. With the proliferation of social media platforms and the exponential growth of user engagement, the need for efficient and scalable content moderation solutions has become increasingly critical. SafeFeed is a platform that harnesses the power of Generative AI and Large Language Models (LLMs) to revolutionize the content moderation process.

SafeFeed is a tool that helps social media platforms manage and filter content based on the situation. It aims to create healthier online communities by making sure that the content shared is appropriate and respectful. By leveraging cutting-edge technologies, including Large Language Models (LLMs) and advanced data processing pipelines, SafeFeed equips platforms with the tools to proactively identify and address potentially harmful content, while preserving the essence of open discussions.

## Ideation

SafeFeed is built on the idea of making content moderation accessible to everyone. It allows communities to manage their online spaces according to their own policies and rules. The platform offers a variety of tools such as automated content checking, dashboards for metrics, policy customization, and active monitoring to ensure a safe environment.

SafeFeed's inception was driven by the realization that traditional content moderation methods, often relying on manual interventions, are increasingly strained by the sheer volume of user-generated content. This recognition fueled the exploration of Generative AI and LLM technologies as a means to augment and streamline the moderation process, ultimately leading to the development of SafeFeed.

# Scraping Reddit Policies for chatbot and submission integrity checks

## 1. Function: scrape\_reddit\_policies

```
def scrape_reddit_policies():  
    """  
    Scrape content policies from Reddit's official policy page and extract hyperlinks, headings, paragraphs, and list items.  
    It first fetches the web page, parses links and text content, and then uses Selenium to navigate to each link to extract additional contents.  
    The extracted contents are stored in a dictionary and eventually written to a CSV file named 'reddit_policies.csv'.  
  
    Side Effects:  
    - Sends HTTP requests to Reddit's content policy page.  
    - Initiates a Selenium WebDriver to navigate and scrape contents of linked web pages.  
    - Creates and writes to 'reddit_policies.csv' in the working directory.  
  
    Returns:  
    None  
    """
```

### Description

This function is designed to automate the collection of content policy information from Reddit's official policy page. It performs the following key tasks:

1. **HTTP Request:** Sends an HTTP request to "<https://www.redditinc.com/policies/content-policy>" using a custom user-agent to fetch the webpage.
2. **Content Parsing:** Utilizes BeautifulSoup to parse the HTML content of the page. Extracts and stores all hyperlinks found within the content section in a dictionary for further use.
3. **Text Extraction:** Identifies and concatenates text from headers (h1, h2, h3), paragraphs (p), and list items (li) to compile a comprehensive text representation of the webpage's content.
4. **Deep Scraping with Selenium:** For each hyperlink extracted, the function uses Selenium WebDriver to navigate to these links. It waits for the page to load, extracts the main content from each linked page, and stores it in a dictionary keyed by the link text.
5. **Output Handling:** Aggregates all extracted content into a single dictionary and outputs the details into a CSV file named 'reddit\_policies.csv'. Each entry in the CSV file represents a key-value pair of link text and its corresponding content.

### Side Effects

- HTTP requests are made to an external website, which involves network communication and can be impacted by network-related issues such as latency or connection errors.
- A browser window is controlled programmatically to scrape dynamic content, which requires appropriate WebDriver and browser installations.

- A CSV file is generated and saved to the local file system, which requires write permissions in the running directory.

### Parameters

- None

### Returns

- None: This function does not return any value but outputs a CSV file containing scraped data.

## Processing (Trials)

### 1. Function: vectorize\_policies

```
def vectorize_policies():  
  
    """  
    Connects to a Snowflake database to fetch all records from a specified table, then vectorizes the text data using the  
    SentenceTransformer model. The vectors are stored in a Pinecone vector database for similarity searching.  
  
    The function assumes the Snowflake table contains a 'key' and 'value' column where 'key' is unique and 'value' contains  
    the text to be vectorized. It handles connections, error management, and closing connections safely. After fetching  
    and processing the data, it initializes a Pinecone index and upserts the vectors into this index.  
  
    Side Effects:  
    - Makes network requests to Snowflake and Pinecone services.  
    - Prints errors and other operational messages to the console.  
    - Modifies data in Pinecone's vector database.  
  
    Returns:  
    None  
    """
```

### Description

This function is designed to vectorize policy texts fetched from a Snowflake database and store these vectors in Pinecone, a vector database, for future similarity searching and retrieval. The process involves several key steps:

1. Database Connection: Establishes a connection to Snowflake using helper functions (not shown here) and fetches all records from a specified table.
2. Data Retrieval: Executes a SQL query to fetch 'key' and 'value' pairs from the table, storing them in a local dictionary for processing.
3. Error Handling: Includes robust error handling during database operations, ensuring the database connection is safely closed regardless of whether the operation was successful or not.

4. Text Vectorization: Utilizes the 'SentenceTransformer' model to convert text data into numerical vectors. The specific model used here is 'all-MiniLM-L6-v2', chosen for its efficiency and performance in generating meaningful text embeddings.
5. Vector Database Initialization and Management: Initializes a new index in Pinecone if it doesn't exist, with parameters tailored for text embeddings. Upserts the generated vectors into this index using unique keys from the Snowflake database.
6. Resource Cleanup: Closes the connection to Snowflake and ensures all resources are properly released after operations.

### Side Effects

- Network requests are made to external services, which involves potential data transfer delays and dependency on external API availability.
- Data modifications occur in a Pinecone vector database, reflecting updates in real-time.

### Parameters

- None

### Returns

- None: This function operates by side effects, modifying external databases without returning any values.

## 2. Function: similarity\_search

```
def similarity_search(text):  
  
    """  
    Performs a similarity search on a given text by vectorizing it and querying a Pinecone index named 'reddit-policies'.  
    After retrieving the top matches from Pinecone, it fetches the corresponding textual data from a Snowflake database.  
  
    The function assumes there is a Pinecone index already set up and a Snowflake table ready to be queried, both containing  
    relevant data. It integrates text embedding and vector search to find and return the most similar text entries based on  
    cosine similarity.  
  
    Args:  
        text (str): The text to search for similarities.  
  
    Returns:  
        dict: A dictionary of keys and values representing the IDs and textual content of the top matches found in Pinecone  
        and fetched from Snowflake.  
  
    Side Effects:  
        - Makes network requests to Pinecone and Snowflake services.  
        - Prints matches and errors to the console.  
    """
```

## Description

The **similarity\_search** function is designed to find text entries similar to a given input text. This is achieved through a series of steps involving both Pinecone and Snowflake databases. The process includes:

1. **Text Vectorization:** Converts the input text into a numerical vector using the SentenceTransformer model 'all-MiniLM-L6-v2'.
2. **Similarity Search in Pinecone:** Queries a pre-configured Pinecone index named 'reddit-policies' with the vector to retrieve the top 6 similar items based on cosine similarity.
3. **Fetching Matched Data from Snowflake:** After retrieving match IDs from Pinecone, the function queries a Snowflake table to fetch the corresponding text entries for each ID, thus providing the full textual content of similar entries.

## Parameters

- **text** (str): The text for which similar items are to be searched.

## Returns

- **dict:** Returns a dictionary where keys are the IDs of the matches and values are the corresponding text values fetched from Snowflake.

## Side Effects

- Makes network requests to Pinecone for vector searches and to Snowflake for retrieving text data.
- The console outputs include the similarity matches and any potential errors encountered during the operations.

## 3. Function: check\_policies

```
def check_policies(text, policies):
    """
    Analyzes a given text to determine if it violates specified platform policies using OpenAI's GPT model.
    This function constructs a prompt for the AI model, integrating platform policies and the text to be analyzed,
    and asks the model to decide if the text violates any of the policies with a true or false response along with a
    reason for any violations.

    Args:
        text (str): The social media post title or text to analyze.
        policies (dict): A dictionary of policy keys and their descriptions.

    Returns:
        str: The AI's judgment about the text's compliance with the policies, including a decision of "True" or "False"
        and a reason for the decision if applicable.

    Side Effects:
        - Makes a network request to OpenAI's servers to process the prompt and generate a response.
    """
```

## Description

The **check\_policies** function evaluates whether a given text, such as a social media post or article title, adheres to or violates specific platform policies. This is accomplished by leveraging OpenAI's GPT model to analyze the text against the policies provided.

## Workflow

1. **Initialization:** Loads the OpenAI API key from an environment variable and initializes the OpenAI client.
2. **Prompt Construction:** Combines all platform policy entries into a single formatted string and constructs a detailed prompt for the AI. The prompt includes the platform policies and the post text, directing the AI to strictly determine the adherence to these policies.
3. **AI Analysis:** Sends the constructed prompt to the OpenAI GPT model, which processes the information and returns a judgment on whether the text violates the policies, including a detailed reason if applicable.
4. **Output Handling:** Parses the AI's response, extracting the compliance judgment and the reasoning behind it.

## Parameters

- **text** (str): The text of the social media post or article to be analyzed.
- **policies** (dict): A dictionary where each key-value pair represents a policy identifier and its corresponding description.

## Returns

- **str:** A formatted response from the AI indicating whether the post violates any policies ("True" or "False") and an explanation of the reasons for any violations.

## Side Effects



- Engages in network communication with OpenAI's API, depending on the availability and response of the external service.

# Processing (Actual)

## Policy Checker Assistant

ASSISTANT

asst\_ZXhmxvnuXjjIIgYsoqcmoEbX

Playground ↗

Name

Policy checker

asst\_ZXhmxvnuXjjIIgYsoqcmoEbX

Instructions

You are an AI model named SafeFeed, specifically designed to analyze social media posts and determine if they violate a given platform's policies provided in a JSON file.

Your responses should follow this format:

Model

gpt-4-turbo-2024-04-09

TOOLS

☒ File search ⓘ

+ Files

Untitled storage

vs\_Yf52DxY7KVZQwmesU8oDz5D2

36 KB

☐ Code interpreter ⓘ

+ Files

Functions ⓘ

+ Functions

MODEL CONFIGURATION

Response format

☐ JSON object ⓘ

Temperature

0.03

Top P

1

API VERSION

Latest ⓘ

Switch to v1

# OpenAI Assistant Overview

**Assistant Name:** Policy Checker

**Assistant ID:** asst\_ZXhmxvnuXjjTlgYsoqcmoEbX

## Model Configuration:

- **Model:** gpt-4-turbo-2024-04-09
- **Response Format:** JSON object
- **Temperature:** Set to **0.03** for factual responses
- **Top P:** Set to **1** to enable the model to use the full range of possible continuations

## Instructions:

The assistant, named SafeFeed, is instructed to strictly analyze social media posts to determine if they violate the given platform's policies. It should respond with either "True" or "False" to indicate a policy violation. If "True," the assistant is expected to provide a detailed reason for the violation. If "False," no further explanation is required.

## Tools:

- **File Search:** The assistant has access to a tool for searching files, which can potentially be used to query documents or databases storing platform policies.
- **Code Interpreter:** This tool allows the assistant to run code, likely enabling it to execute scripts that could aid in analysis or automated decision-making processes.
- **Functions:** The assistant has the capability to perform functions, which are pre-defined operations or scripts, to assist with its tasks.

**API Version:** Latest

## Usage:

The assistant seems to be used in a policy enforcement or moderation system where the content of posts needs to be checked against a set of rules or guidelines. It can be integrated into a moderation pipeline to automatically flag posts that may require human review or direct action, such as removal or tagging for sensitivity.

## Applications:

- Content Moderation on Social Platforms
- Compliance Checking in User-Generated Content
- Automated Enforcement of Community Guidelines

## Operational Notes:

- The assistant is part of a more extensive system with a defined workflow for policy checks.

- It interfaces with other services or databases where the actual policy documents are stored and managed.

#### Data Handling:

- The assistant is configured to use files stored within OpenAI, containing the policies needed for the decision-making process.

## Detailed Function Descriptions

### 1. Function: list\_files

```
def list_files():  
    """  
    Retrieves a list of files stored in the OpenAI platform and returns them as a dictionary where keys are filenames  
    and values are the corresponding file IDs.  
  
    Returns:  
    | dict: A dictionary mapping filenames to their respective file IDs.  
    """
```

#### Description

This function interacts with the OpenAI API to retrieve a list of all files stored under the user's account. It formats this list into a dictionary, making it easier to access any file by its name.

#### Workflow

1. **Retrieve Files:** Makes an API call to list all files stored in the OpenAI platform.
2. **Format Response:** Converts the list of files into a dictionary where each key-value pair represents a filename and its corresponding file ID.

#### Returns

- **dict:** A dictionary where filenames are keys, and their corresponding file IDs are values.

### 2. Function: create\_thread

```
def create_thread():  
    """  
    Creates a new conversation thread in the OpenAI API environment. This thread is used for managing and maintaining  
    the state of conversations over time.  
  
    Returns:  
    | str: The unique identifier for the newly created thread.  
    """
```

## Description

Creates a new conversation thread via the OpenAI API. Each thread is an independent interaction session, allowing the user to manage conversations distinctly. This is particularly useful in applications involving continuous dialogues or multiple users.

## Workflow

1. **Thread Creation:** Sends a request to OpenAI to create a new thread.
2. **Return Thread ID:** Retrieves the ID of the newly created thread and returns it.

## Returns

- **str:** The ID of the newly created thread, which can be used to manage conversation flows within that thread.

## 3. Function: send\_message

```
def send_message(thread_id, user_input, file_id):  
  
    """  
    Sends a user input as a message to a specified thread in the OpenAI API environment, attaching a file if provided.  
  
    Args:  
        thread_id (str): The unique identifier of the thread to which the message will be sent.  
        user_input (str): The text content of the message to be sent.  
        file_id (str): The identifier of the file to attach to the message.  
  
    Returns:  
        OpenAI.ThreadMessage: The message object as returned by the OpenAI API.  
    """
```

## Description

This function is responsible for sending a message within a specified conversation thread managed by OpenAI's API. The message can include textual content and optionally a file attachment.

## Workflow

1. **Message Creation:** Constructs and sends a message to a specified thread using OpenAI's API. The message contains user-provided text and may include an attached file identified by **file\_id**.
2. **Return Message Object:** Captures and returns the message object as generated by the API, including details like message ID, timestamp, and any associated metadata.

## Parameters

- **thread\_id (str):** ID of the thread to which the message is sent.

- **user\_input** (str): Text content of the message.
- **file\_id** (str): ID of the file to be attached to the message.

#### Returns

- **OpenAI.ThreadMessage**: An object representing the message that was sent.

## 4. Function: wait\_on\_run

```
def wait_on_run(run, thread_id):
    """
    Monitors the status of a running operation within a thread until it is completed. Polls the operation's status
    at regular intervals and returns the run object once the status is no longer 'queued' or 'in_progress'.

    Args:
        run (OpenAI.Run): The initial run object to monitor.
        thread_id (str): The unique identifier of the thread associated with the run.

    Returns:
        OpenAI.Run: The final run object after completion.
    """
```

#### Description

Monitors the status of an operation (run) in an OpenAI thread. It continuously checks the status until the operation completes, ensuring that any subsequent actions only proceed once the current operation has fully resolved.

#### Workflow

1. **Status Check:** Repeatedly checks the run's status at half-second intervals.
2. **Completion Detection:** Continues the checks until the run's status changes from 'queued' or 'in\_progress' to a completed state.

#### Parameters

- **run** (OpenAI.Run): The run object to monitor.
- **thread\_id** (str): The thread ID associated with the run.

#### Returns

- **OpenAI.Run**: The updated run object reflecting the completion status.

## 5. Function: analyze\_content\_image

```
def analyze_content_image(thread_id, assistant_id, user_input, file_id):
    """
    Sends a user input along with an image file to an OpenAI API thread, and analyzes whether the content and image
    violate specified platform policies. The analysis includes detailed assessments of the content's compliance and
    characteristics of the image (general, sensitive, explicit).

    Args:
        thread_id (str): The unique identifier of the thread to which the analysis request is sent.
        assistant_id (str): The identifier of the OpenAI Assistant to use for processing the request.
        user_input (str): The text content associated with the social media post to be analyzed.
        file_id (str): The identifier of the image file associated with the post.

    Returns:
        OpenAI.Run: The run object containing the AI's response to the policy violation analysis.

    This function orchestrates sending the message, creating a run, and waiting for the run to complete, thereby
    providing a comprehensive analysis of the text and image in relation to platform policies.
    """
```

## Description

This function integrates with the OpenAI API to provide a detailed analysis of social media content and associated images against specified platform policies. It evaluates if the content violates any policies and characterizes the image based on predefined criteria (general, sensitive, explicit).

## Workflow

1. **Send Message:** Initiates the process by sending the user input and the image file to the specified thread using the **send\_message** function.
2. **Create Run:** Creates a run within the thread using a specific assistant. This run contains detailed instructions for the AI to analyze both the text and image content.
3. **Wait for Run Completion:** Monitors the run's progress until completion, ensuring that all analyses are finished before proceeding.
4. **Return Analysis:** Returns the completed run, which includes the AI's detailed analysis and conclusions regarding the content and image.

## Parameters

- **thread\_id** (str): ID of the thread to which the message and analysis request are sent.
- **assistant\_id** (str): ID of the OpenAI Assistant tasked with analyzing the content.
- **user\_input** (str): The text content of the social media post.
- **file\_id** (str): The ID of the image file associated with the post.

## Returns

- **OpenAI.Run:** Contains the detailed response from the AI, including:
  - **Policy Violation: True or False**

- **Reason for Violation: Text** explaining the reason if there is a violation.
- **Is image general: True or False**
- **Is image sensitive: True or False**
- **Is image explicit: True or False**
- **Should it be Removed: True or False**
- **Are you sure about this decision: True or False**

## 6. Function: analyze\_content\_text

```
def analyze_content_text(thread_id, assistant_id, user_input, file_id):
    """
    Analyzes a social media post's text to determine if it violates specified platform policies using an AI model.
    The function sends the text for analysis, waits for the AI to process it, and returns the analysis, which includes
    decisions on policy violation, reasons for any violations, and recommendations on content removal.

    Args:
        thread_id (str): The unique identifier of the thread where the analysis is performed.
        assistant_id (str): The identifier of the OpenAI Assistant to use for processing the request.
        user_input (str): The text of the social media post to be analyzed.
        file_id (str): The identifier of an optional file associated with the post.

    Returns:
        OpenAI.Run: The run object containing the AI's response to the policy violation analysis.

    The response format from the AI is expected to strictly adhere to defined output fields regarding policy violations,
    reasons, and removal recommendations.
    """
```

### Description

This function is designed to analyze text content of social media posts to ascertain if they comply with or violate specific platform policies. It employs an OpenAI Assistant, which is directed to assess the content based on predefined instructions. The function manages the entire process from sending the initial message to returning the completed analysis.

### Workflow

1. **Send Initial Message:** Uses the **send\_message** function to send the user input along with any associated file to a specific thread.
2. **Create Run for Analysis:** Initiates a run with detailed instructions for the AI to evaluate the text for policy violations.
3. **Wait for Run Completion:** Monitors the run's status, ensuring that the analysis is completed before proceeding.



4. **Return Analysis Results:** Returns the run object, which includes the AI's conclusions on policy violations, reasons for violations if any, and content removal advice.

#### Parameters

- **thread\_id** (str): ID of the thread in which the analysis is to be performed.
- **assistant\_id** (str): ID of the OpenAI Assistant tasked with the analysis.
- **user\_input** (str): The text of the social media post.
- **file\_id** (str): ID of the file related to the post, if applicable.

#### Returns

- **OpenAI.Run:** Contains the detailed response from the AI, including:
  - **Policy Violation:** Indicates whether the post violates any policy (True/False).
  - **Reason for Violation:** Provides details on why the post is considered a violation.
  - **Should it be Removed:** Advises whether the post should be removed (True/False).
  - **Are you sure about this decision:** Affirms the confidence level of the AI's decision (True/False).

## 7. Function: get\_responses

```
def get_responses(thread_id):  
    """  
    Retrieves all responses from the assistant in a specified OpenAI API thread. This function lists all messages  
    within a thread and filters out responses that were generated by the assistant.  
  
    Args:  
        thread_id (str): The unique identifier of the thread from which to fetch assistant responses.  
  
    Returns:  
        list: A list of text responses from the assistant.  
    """
```

#### Description

This function is designed to retrieve all responses from an OpenAI assistant within a specified conversation thread. It filters and returns only those messages that are responses from the assistant, providing a clear view of how the assistant has interacted within the thread.

#### Workflow

1. **Fetch Messages:** Connects to the OpenAI API and retrieves a complete list of messages from a specified thread.

2. **Filter Responses:** Filters the messages to include only those where the role is marked as "assistant," ensuring that only responses generated by the AI are considered.
3. **Extract Content:** Extracts the text content from each assistant message and compiles them into a list.

### Parameters

- **thread\_id** (str): The ID of the thread from which to retrieve the assistant's responses.

### Returns

- **list:** A list containing the text of each response made by the assistant in the thread.

## 8. Function: extract\_information\_image

```
def extract_information_image(response):  
    """  
    Extracts specific information from a text response using regular expressions. This function is designed to parse a  
    structured text response to identify and extract key details such as policy violation status, reasons for violation,  
    and various attributes related to image analysis.  
  
    Args:  
        response (str): The text response from which information is to be extracted.  
  
    Returns:  
        dict: A dictionary containing the extracted values for each key, such as policy violation status, reason for  
        violation, image characteristics, and removal recommendation. If a particular piece of information is not found,  
        the corresponding value will be None.  
  
    This function uses predefined regular expressions to search for specific patterns related to content policy  
    violations and image assessments in the response text.  
    """
```

### Description

This function parses a structured text response to systematically extract relevant information using regular expressions. The information extracted includes policy violation decisions, detailed reasons for violations, and attributes related to image content analysis such as generality, sensitivity, explicitness, and removal recommendation.

### Workflow

1. **Define Patterns:** Sets up a dictionary of regular expression patterns corresponding to different information segments within the response text.
2. **Pattern Matching:** Iterates through each pattern, applying it to the response text to find matches.

3. **Extract and Store Results:** Each found match is extracted and stored in a results dictionary under its respective key. If no match is found for a particular pattern, the key's value is set to None.

### Parameters

- **response** (str): The structured text response containing the data to be extracted.

### Returns

- **dict:** A dictionary where each key corresponds to an information segment (e.g., **policy\_violation**, **reason\_for\_violation**), and each value is the extracted data or None if the pattern was not matched.

## 9. Function: extract\_information\_text

```
def extract_information_text(response):  
  
    """  
    Extracts key information from a structured text response using regular expressions, focused on analyzing text-based  
    social media posts for policy compliance. The function identifies whether the content violates policies and the  
    rationale behind such decisions, as well as recommendations on content removal and certainty of the decision.  
  
    Args:  
        response (str): The text response from which information is to be extracted, formatted according to specific  
        patterns indicating policy compliance outcomes.  
  
    Returns:  
        dict: A dictionary containing the extracted values for key details such as policy violation status, reasons for  
        violation, removal recommendation, and decision certainty. If a particular piece of information is not found,  
        the corresponding value will be set to None.  
  
    This function is tailored to parse responses that conform to a specific response format, ensuring precise extraction  
    of required information for effective decision-making in content moderation.  
    """
```

### Description

This function is designed to parse structured text responses to extract predefined information related to policy violations in text-based content. It efficiently identifies and extracts details about policy violations, reasons for these violations, and associated moderation recommendations.

### Workflow

1. **Define Patterns:** Establishes a set of regular expression patterns corresponding to different segments of information within the response text.
2. **Apply Patterns and Extract Data:** Iterates through each pattern to apply it to the response text, extracting matches and storing them in a results dictionary.
3. **Handle Non-matches:** Ensures that information segments without matches are represented in the results dictionary with a value of None.

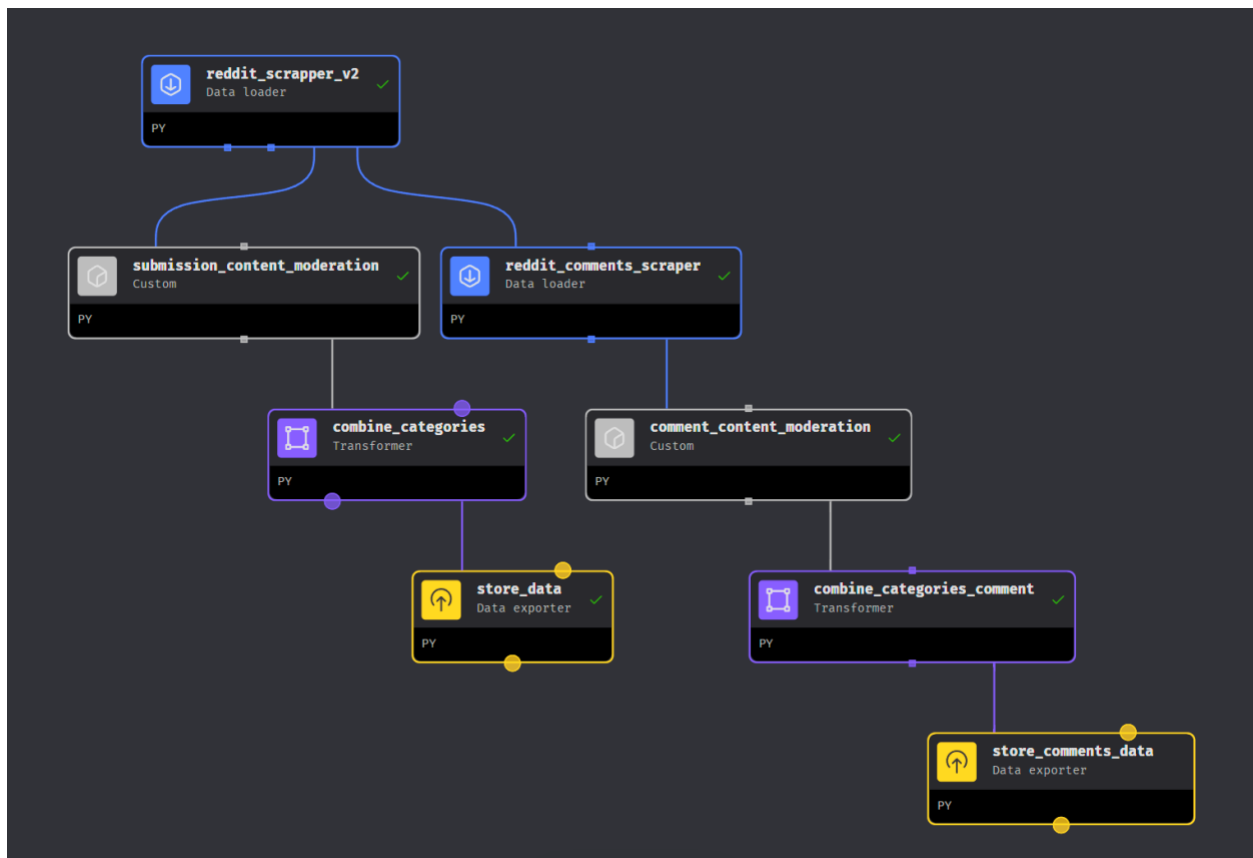
## Parameters

- **response** (str): The response text that includes structured information about content policy compliance.

## Returns

- **dict**: A dictionary with keys representing information segments (e.g., **policy\_violation**, **reason\_for\_violation**) and values holding the extracted data or None for unmatched patterns.

## Detailed Workflow



### 1. reddit\_scrapper\_v2 - Data Loader

## Overview

**reddit\_scrapper\_v2** is a Python-based data loader designed to extract submissions from specified subreddits using the Reddit API, represented by PRAW (Python Reddit API Wrapper). It preprocesses text data, evaluates sentiment, integrates with OpenAI for content moderation, and incorporates image recognition capabilities for enhanced content analysis.

### Functionalities

- **API Integration:** Connects to Reddit using credentials fetched from environment variables to securely access subreddit data.
- **Sentiment Analysis:** Utilizes the VADER Sentiment Analysis tool to categorize post sentiment as Positive, Negative, or Neutral.
- **Content Moderation:** Leverages OpenAI's content moderation capabilities to assess posts against specified categories such as harassment, hate speech, or threats.
- **Image Processing:** Extracts image URLs from posts and predicts image content tags using a Gradio client interface, potentially linking to a machine learning model.
- **Data Handling:** Gathers post attributes and sentiment into a DataFrame structure, preparing it for storage or further analysis.
- **Timestamp Management:** Updates the last processed timestamp for each subreddit to prevent reprocessing of data.
- **Snowflake Connection:** Interfaces with Snowflake, a cloud data platform, to retrieve and update subreddit information.
- **Data Testing:** Implements a testing routine to ensure data integrity and structure.

### Usage

This loader is used in contexts where comprehensive data extraction and preprocessing from Reddit are required. It is particularly useful for sentiment analysis, content moderation, and machine learning tasks in social media analytics.

## 2. submission\_content\_moderation - Custom Transformer

### Overview

The **submission\_content\_moderation** block is a custom Python transformer within the data pipeline that analyzes Reddit submissions for compliance with community guidelines. It leverages sentiment analysis, OpenAI's moderation API, and an image tag prediction model to determine if a submission should be moderated.

### Functionalities

- **Preprocessing:** Textual content from Reddit submissions is cleaned and preprocessed, removing URLs, punctuation, and translating emojis into text.
- **Sentiment Analysis:** Determines the sentiment of each submission using the VADER Sentiment Analysis tool.
- **Content Moderation:** Integrates with OpenAI's moderation API to classify the content of submissions into various moderation categories.
- **Image Tagging:** Processes image URLs within submissions to predict tags and assess content suitability.
- **Policy Violation and Deletion:** Assesses if the submissions violate any policies and marks them for deletion, following specific conditions.

### Moderation and Deletion Logic

- If a submission is flagged by OpenAI's moderation API or contains image captions that suggest content moderation, it triggers a response process.
- The process retrieves policy documents, creates a thread for analysis, and gathers OpenAI's moderation responses.
- Based on the analysis, the block assigns categories indicating potential policy violations and if the submission should be deleted.

### Usage

- The transformer is employed in scenarios where automated, AI-driven content moderation is needed to manage and enforce community standards on Reddit.
- It is especially useful in high-traffic subreddits where manual moderation may not be feasible due to the volume of submissions.

## 3. reddit\_comments\_scraper - Data Loader

### Overview

The **reddit\_comments\_scraper** is a custom data loader implemented in Python to retrieve and process comments from Reddit submissions. It connects to Reddit via PRAW and Snowflake for database operations, carrying out sentiment analysis and content moderation assessment on comments.

### Functionalities

- **Snowflake Connection:** Establishes a connection to the Snowflake database to retrieve submission records and update timestamps after processing.

- **Reddit API Interaction:** Utilizes PRAW to fetch comments from Reddit submissions based on IDs obtained from Snowflake.
- **Comment Processing:** Iterates through comments, performing text preprocessing, sentiment analysis using VADER, and content moderation using OpenAI's API.
- **Recursive Comment Retrieval:** Implements a function to recursively gather data from nested comment threads, preserving the hierarchical structure and relational context.
- **Data Structuring:** Organizes comment data into a pandas DataFrame, enabling easy manipulation and export for further analysis or storage.
- **Timestamp Management:** Updates the 'LAST\_TRIGGER\_TIMESTAMP' in Snowflake for each submission, ensuring comments are processed from the last known point.

### Usage

- This loader is primarily used for extracting comments from Reddit submissions in real-time, processing them for compliance with community guidelines, and storing them for future review or analytics.

## 4. combine\_categories – Transformer

### Overview

The **combine\_categories** transformer is a Python function that consolidates multiple content moderation categories into aggregated columns within a DataFrame. This transformation simplifies the dataset by combining related flags into broader categories that indicate the presence of specific types of content violations in text.

### Functionalities

- **Category Aggregation:** This function creates new binary columns in the DataFrame that represent whether any form of hate speech, harassment, self-harm, sexual content, or violence is detected in the text.
- **Column Reduction:** Original columns that represented individual moderation categories are removed after aggregation to streamline the dataset and focus on the newly created summary indicators.
- **Conditional Logic:** Applies logical OR operations across related columns to determine the aggregated category flags, ensuring that if any related flags are true, the aggregated column reflects it.

### Usage

- This transformer is typically employed post-moderation processing to simplify datasets for easier interpretation and analysis.

- The aggregated flags can be used for high-level reporting or to trigger content moderation actions within a larger automated system.

## 5. comment\_content\_moderation - Transformer

### Overview

The **comment\_content\_moderation** block serves as a custom Python transformer that performs content moderation actions on Reddit comments. It evaluates comments for policy violations and, if flagged, triggers moderation processes such as comment removal and user notification.

### Functionalities

- **Moderation Logic:** Iterates through each comment's data within the DataFrame, checking for flags that indicate policy violations.
- **Flag Management:** Updates the DataFrame to reflect the moderation status of each comment, marking them as deleted if flagged.
- **Error Handling:** Captures exceptions that may arise during the moderation process, ensuring the system remains robust and functional even when facing individual operation failures.

### Testing and Assurance

- The function includes a test decorator **@test** to define test cases that assert the presence and validity of the output DataFrame.
- This testing strategy helps ensure that the moderation logic works as expected and that the DataFrame's integrity is preserved post-transformation.

### Usage

- This transformer is crucial for automated moderation workflows on platforms like Reddit, where it can autonomously enforce community guidelines and manage user-generated content at scale.
- The intended moderation actions illustrate a potential application where automated systems can take direct action to maintain the standards of digital communities.



## 6. store\_data - Data Exporter

### Overview

The **store\_data** block is a Python function annotated as a **data\_exporter** that facilitates the export of processed data, contained in a pandas DataFrame, into a Snowflake database table.

### Functionalities

- **Database Connection:** Establishes a connection to the Snowflake database using the provided credentials.
- **Table Management:** Ensures the existence of the designated schema and table, preparing the database for data ingestion.
- **Data Export:** Utilizes a Snowflake loader to append the processed DataFrame (**df\_posts**) to the specified table within the Snowflake database.
- **Config Management:** Leverages a configuration file, specified by **config\_path**, to manage export settings.
- **Exception Handling:** Implements error handling for any issues that arise during the connection or data export process.
- **Resource Cleanup:** Ensures that database resources, such as cursor and connection, are closed after the operation to prevent leaks.

### Usage

- This exporter is used to store the output of content moderation, sentiment analysis, and other transformations, into a structured and queryable format within Snowflake.
- The **if\_exists='append'** parameter indicates that new data is added to the existing table, supporting incremental updates.

## 7. combine\_categories\_comment – Transformer

### Overview

**combine\_categories\_comment** is a Python transformer designed to consolidate individual content moderation category flags into broader indicators of content type within a DataFrame. This transformer simplifies downstream analysis and decision-making processes by summarizing the various content moderation signals into clear, binary columns.

### Functionalities

- **Data Transformation:** Introduces new binary columns to represent aggregated categories of content moderation concerns, such as hate speech, harassment, self-harm, sexual content, and violence.
- **Logical Aggregation:** Applies logical operations across related category flags to produce aggregated indicators. For example, if any related flags for hate speech are true, the **IS\_TEXT\_HATE\_SPEECH** column will be marked as true.
- **Column Reduction:** Drops the individual content moderation category columns after their values have been aggregated, reducing the DataFrame's complexity and focusing on the newly created summary columns.

### Usage

- This transformer is utilized within a data pipeline to streamline the content moderation data structure, making it more efficient for reporting, monitoring, or triggering automated actions based on the content type.
- The output is a cleaned DataFrame with comprehensive flags indicating the presence of various types of content that may require attention or action.

## 8. store\_comments\_data - Data Exporter

### Overview

The **store\_comments\_data** function, marked as a **data\_exporter**, is tasked with uploading a DataFrame of processed Reddit comments into a designated Snowflake table. This process ensures that comment data undergoes long-term storage for analytics or moderation follow-up actions.

### Functionalities

- **Snowflake Connection:** Initiates a connection to the Snowflake data warehouse using secure authentication details.
- **Data Export Operation:** The function assumes an established table structure within Snowflake and appends the new comment data to this table.
- **Config File Usage:** Employs an external configuration file to manage database connection properties and export settings.
- **Error Handling:** Captures and logs any exceptions encountered during the database operations to alert to issues without disrupting the entire process.
- **Resource Management:** Ensures the proper closing and cleanup of database resources after the export operation is completed to maintain system stability and integrity.

### Usage

- The exporter is used after comments have been extracted, processed, and are ready to be saved into Snowflake, making the data accessible for future queries and reporting.
- The **if\_exists='append'** parameter in the data export function indicates that new data will be added to the table, facilitating incremental updates.

# Policy guide Chatbot

## Policy guide assistant

**ASSISTANT**  
**asst\_FWgfzg1xu8p3NLV0rzBjAZBk**[Playground ↗](#)

**Name**  

Policy guide

asst\_FWgfzg1xu8p3NLV0rzBjAZBk

**Instructions**  

You are an AI assistant with access to a set of policies provided in an attached JSON file. Your task is to understand these policies thoroughly and answer users' questions based on this information. When answering, please be brief and directly address the user's query. If applicable, include a relevant chunk or key points from the policies that directly relate to the user's question. Your responses should help users understand how their inquiries relate to the

**Model**  

gpt-3.5-turbo

**TOOLS**

☒ **File search** ⓘ[+ Files](#)

Vector store for Policy guide  
vs\_SQ1KpW59GxtSp6dxCbcsLoa4

36 KB

☐ **Code interpreter** ⓘ[+ Files](#)

**Functions** ⓘ[+ Functions](#)

**MODEL CONFIGURATION**

**Response format**  
☐ JSON object ⓘ

Temperature

11

Top P

1

**API VERSION**  

Latest ⓘ[Switch to v1](#)

# OpenAI Assistant Overview

**Assistant Name:** Policy guide

**Assistant ID:** asst\_FWgfzg1xu8p3NLV0rzBjAZBk

## Model Configuration:

- **Model:** gpt-3.5-turbo
- **Response Format:** JSON object
- **Temperature:** Set to **1** for creative responses
- **Top P:** Set to **1** to enable the model to use the full range of possible continuations

## Instructions:

The assistant is instructed to interpret and relay information from policy documents to answer users' queries. It must provide concise, relevant information from the policies that directly addresses each user's question, aiding in their understanding of how the queries relate to specific policy details. The goal is to offer clear and precise responses without requiring users to review the entire policy document.

## Tools:

- **File Search:** The assistant has access to a tool for searching files, which can potentially be used to query documents or databases storing platform policies.

**API Version:** Latest

## Usage:

The AI assistant is designed to provide quick and accurate explanations of policy details in response to user inquiries. It uses policy documents to inform its answers, ensuring they are precise and relevant to the specific questions asked. The assistant's responses are meant to be clear and concise, focusing on the essential aspects of the policies related to the user's questions, thereby helping users understand the policies without having to read them in full.

## Applications:

- Summarization of policies
- Compliance Checking in User-Generated Content

## Operational Notes:

- The assistant is part of a more extensive system with a defined workflow for policy guidance.
- It interfaces with other services or databases where the actual policy documents are stored and managed.

## Data Handling:

- The assistant is configured to use files stored within OpenAI, containing the policies needed for the decision-making process.

## Overview

This Streamlit application integrates with the OpenAI API to provide a conversational AI assistant that helps users understand and navigate policy documents. The app leverages OpenAI's capabilities to analyze policy-related questions posed by users and responds with relevant excerpts or insights derived from the policies.

## Key Components

1. **Initialization and Environment Setup:** The app begins by importing necessary packages and loading environment variables, particularly the OpenAI API key. It initializes global variables for API keys, assistant IDs, and file references which are necessary for subsequent API calls.
2. **Streamlit Cache and OpenAI Client Setup:** Utilizes Streamlit's caching mechanism to load and persist the OpenAI client and the assistant across sessions, ensuring efficient resource usage.
3. **Assistant Thread Creation:** Sets up an assistant thread using OpenAI's API to handle ongoing conversations without needing to reinitialize context or settings.
4. **User Interaction and Message Handling:** Implements a user interface to accept input and displays responses using Streamlit widgets. The user inputs are processed, and responses are generated by the OpenAI assistant based on the policy documents.
5. **Conversation Management:** Manages a conversation history within the session state to allow for a continuous interaction flow. This history includes both user queries and AI responses, displayed in a conversational format.

## Detailed Function Descriptions

1. Function: `load_openai_client_and_assistant`

```

@st.cache_resource
def load_openai_client_and_assistant():
    """
    Initializes and retrieves the OpenAI client and a specific assistant using provided API keys and assistant ID.
    This function is decorated with `st.cache_resource` to cache the results, reducing the number of API calls by
    reusing the client and assistant objects across sessions unless the cache is cleared or invalidated.

    Returns:
        tuple: Returns a tuple containing the OpenAI client and the assistant object.
        This allows for efficient reuse in applications, particularly in environments like Streamlit where
        repeated API calls can slow down interactions.
    """

```

## Overview

This function is crucial for initializing connections to OpenAI services, specifically to fetch and reuse the OpenAI client and a designated assistant. It's designed to enhance efficiency in Streamlit applications by caching these objects, thus optimizing API usage and improving application responsiveness.

## Description

The function initializes an OpenAI client using an API key stored in the environment or provided globally within the code. It then retrieves a specific assistant identified by an `assistant_id`. The `st.cache_resource` decorator is used to cache these objects, preventing unnecessary reinitializations and reducing load times for subsequent function calls.

## Returns

- **tuple:** A tuple containing two elements:
  - **OpenAI.Client:** The initialized OpenAI client, configured and ready to make requests.
  - **OpenAI.Assistant:** The specific assistant object retrieved using the assistant ID, ready for interaction.

## Streamlit Caching

- The `st.cache_resource` decorator is crucial for applications in Streamlit, enabling the caching of expensive resources. In this case, the expensive operation is the API call made to retrieve the assistant. Caching ensures that once the assistant is retrieved, it is stored and reused without the need to make another API call until the cache expires or is invalidated.

## 2. Function: `wait_on_run`

```
def wait_on_run(run, thread_id):
    """
    Monitors the status of a running operation within a thread until it is completed. Polls the operation's status
    at regular intervals and returns the run object once the status is no longer 'queued' or 'in_progress'.

    Args:
        run (OpenAI.Run): The initial run object to monitor.
        thread_id (str): The unique identifier of the thread associated with the run.

    Returns:
        OpenAI.Run: The final run object after completion.
    """
```

## Description

Monitors the status of an operation (run) in an OpenAI thread. It continuously checks the status until the operation completes, ensuring that any subsequent actions only proceed once the current operation has fully resolved.

## Workflow

3. **Status Check:** Repeatedly checks the run's status at half-second intervals.
4. **Completion Detection:** Continues the checks until the run's status changes from 'queued' or 'in\_progress' to a completed state.

## Parameters

- **run** (OpenAI.Run): The run object to monitor.
- **thread\_id** (str): The thread ID associated with the run.

## Returns

- **OpenAI.Run:** The updated run object reflecting the completion status.

## 3. Function: get\_assistant\_response

```
def get_assistant_response(user_input=""):
    """
    Initiates a response from the OpenAI assistant based on user input, utilizing a specified set of policies provided in
    an attached JSON file. This function handles the interaction with the assistant through the creation of a message and
    a run within an OpenAI thread, followed by waiting for the run to complete to retrieve the assistant's response.

    Args:
        user_input (str): The user's query or statement that needs to be analyzed by the assistant.

    Returns:
        str: The text of the assistant's response based on the analysis of the user input against the provided policies.

    This function ensures the assistant's responses are direct, informative, and relevant to the policies involved, aiding
    users in understanding specific policy details without reading the entire document.
    """
```



## Overview

This function facilitates interaction with an OpenAI assistant to obtain responses to user queries based on predefined policy documents. It sends the user's input to the assistant, along with policy details in a JSON file, and processes the assistant's response to ensure it is informative and directly relevant to the user's inquiry.

## Description

The function executes several steps to get a response from the OpenAI assistant:

1. **File Retrieval:** Fetches the ID of the policy file using a separate function, **get\_file\_id**, which should provide the logic for identifying and retrieving the correct file ID.
2. **Message Creation:** Sends the user input as a message to the assistant within a thread, attaching the policy file for reference.
3. **Run Creation:** Creates a run with specific instructions for the assistant, detailing how to analyze the input and formulate the response.
4. **Wait for Completion:** Monitors the status of the run until completion, ensuring that the response is fully processed before proceeding.
5. **Response Retrieval:** Retrieves all messages posted after the user's initial query and returns the assistant's response.

## Parameters

- **user\_input** (str): Text input from the user which the assistant will analyze in the context of attached policies.

## Returns

- **str:** The assistant's response, which includes analysis and information relevant to the user's input and the attached policies.

## 4. Function: add\_message

```
def add_message(role, content):
    """
    Adds a new message to the chat history stored in the session state. This function is designed to handle the dynamic
    interaction history in a conversational UI, ensuring that each participant's messages are recorded in the order they
    are received.

    Args:
        role (str): The role of the message sender, typically 'user' or 'assistant', indicating who sent the message.
        content (str): The text content of the message to be added to the chat history.

    Effects:
        Modifies the session state by appending a new message to the chat history list. Each message is stored as a
        dictionary with 'role' and 'content' keys.
    """
```

## Overview

This function is an integral part of maintaining a conversation history in applications with interactive interfaces, such as chatbots or customer service tools built with Streamlit. It appends new messages to a session state variable, preserving the flow and context of the conversation.

## Description

The function updates the **chat\_history** list stored in the Streamlit **session\_state**. Each message is added as a dictionary with keys for the sender's role and the message content. This structured approach facilitates easy rendering and management of conversation history in the UI.


## Parameters

- **role** (str): Specifies the sender of the message. Accepted values are typically "user" for the application's user or "assistant" for the automated response system.
- **content** (str): The actual text of the message to be recorded in the chat history.


## Effects

- **Updates Session State:** The function modifies **st.session\_state.chat\_history** by appending a new dictionary object containing the role and message content.

## Detailed Workflow




# Safe-Feed policy guide



Type your message:

Can you summarize all the policies?

Send



## Conversation

You

Can you summarize all the policies?

Assistant

The policies on Reddit cover various aspects including guidelines on content, community interactions, harassment, violence, identity-based attacks, and spam prevention. Here are key points from the policies:

- Content**: Users are expected to follow community rules, respect others' privacy, avoid sharing inappropriate content involving minors, use Reddit with integrity, ensure predictable experiences, and abide by legal guidelines.
- Harassment and Bullying**: Reddit prohibits harassment, threats, and bullying which includes behaviors aimed at intimidating or abusing others, whether in public or private interactions.
- Violence**: Content inciting or glorifying violence, self-harm, or suicide is not allowed. Violent content should be appropriately contextualized, and health misinformation that poses a risk is also prohibited.
- Identity or Vulnerability**: Reddit does not tolerate hate or violence towards marginalized or vulnerable groups based on various characteristics like race, gender, or disability.
- Spamming**: Spamming, defined as unwanted actions negatively affecting users or communities, is against Reddit's policies. Users should post authentic content, avoid irrelevant posting, and not engage in spam-like activities.
- Vote Manipulation**: Utilizing multiple accounts or services to manipulate votes is strictly prohibited.

If you have specific questions about any of these policy areas or need more details on a particular aspect, feel free to ask!

- **Text Input:** Allows users to type their queries.
- **Send Button:** Submits the user's query to the OpenAI assistant.
- **Conversation Display:** Shows the conversation history in a scrollable format, differentiating between user and assistant messages.

### Operational Flow

Upon launching the app, users are presented with a text input field and a send button. Users can type their queries regarding the policies, and upon clicking send, the query is processed by the OpenAI assistant. The assistant accesses policy documents and generates responses that are then displayed to the user. The conversation history is continuously updated and displayed on the screen.

### Usage Example

A user can start a session by entering a question such as "What is the policy on data privacy?" The assistant, after processing the question, will respond with a direct excerpt or a summary based on the attached policy documents, helping the user understand the specific policy details.

## Policies Customizer

### Detailed Function Descriptions

#### 1. Function: openai\_connection

```
def openai_connection():
    """
    Establishes a connection to the OpenAI API using the provided API key.

    Returns:
        OpenAI.Client: Returns an instance of the OpenAI client configured with the specified API key.
    """
```

#### Description

Establishes a connection to the OpenAI API by initializing an OpenAI client with an API key stored in the environment variables. This function simplifies the creation of an OpenAI client instance, which is used in various parts of the application to interact with OpenAI services.

#### Returns

- **OpenAI.Client:** An instance of the OpenAI client, configured to communicate with OpenAI services using the API key.

## 2. Function: initialize\_default\_policies\_and\_folders

```
def initialize_default_policies_and_folders():  
  
    """  
    Initializes the environment by ensuring the existence of necessary policy files and folders. If the file named  
    'reddit_policies.json' does not exist on the OpenAI platform, it uploads the file. Also checks for the presence  
    of a local directory named 'Custom_policies' and creates it if it does not exist.  
  
    Side Effects:  
    - Makes network requests to OpenAI to list and potentially upload files.  
    - Accesses the filesystem to check for directories and possibly create them.  
    - Outputs to console about the creation status of directories.  
  
    Returns:  
    None  
    """
```

### Description

Ensures that necessary infrastructure for policy management is in place. This includes verifying the presence of a specific policy file named 'reddit\_policies.json' on the OpenAI platform and a local directory called 'Custom\_policies'. If the policy file is not found in the OpenAI files list, it is uploaded. The function also checks for the existence of the 'Custom\_policies' directory and creates it if it is not found.

### Workflow

1. **Connect to OpenAI:** Utilizes `openai_connection` to get an OpenAI client.
2. **List and Verify Policy Files:** Lists all files on the OpenAI platform and checks if 'reddit\_policies.json' exists. If it does not exist, the file is uploaded.
3. **Directory Check and Creation:** Checks if the directory 'Custom\_policies' exists locally. If not, it creates the directory.

### Side Effects

- **Network Communication:** Interacts with OpenAI's API to fetch and upload files.
- **Filesystem Modifications:** Checks for and creates directories on the local machine.

### Parameters

- None

### Returns

- None

### 3. Function: load\_policies

```
def load_policies(subreddit_name=None):  
    """  
    Loads policies from a JSON file, attempting to load subreddit-specific policies if a subreddit name is provided.  
    Falls back to default policies if no subreddit-specific file exists.  
  
    Args:  
        subreddit_name (str, optional): The name of the subreddit for which to load specific policies. If None or not provided,  
        the function loads the default policy file.  
  
    Returns:  
        dict or None: Returns a dictionary containing the policies if the file is successfully loaded, or None if the file  
        does not exist or cannot be opened.  
  
    Raises:  
        FileNotFoundError: Raises an exception if no suitable policy file is found at the expected path.  
  
    This function is designed to support dynamic policy loading based on subreddit-specific needs, enhancing flexibility  
    for applications requiring tailored content moderation or policy enforcement.  
    """
```

#### Overview

This function is designed to dynamically load policy documents from JSON files based on subreddit-specific needs or default configurations. It enables applications to tailor responses or actions based on customized policy sets, thereby supporting diverse operational requirements across different subreddit communities.

#### Description

The function tries to load a JSON file containing policies. It first checks if a subreddit-specific file is provided and exists; if not, it loads a default policy file. This functionality supports applications that require specific policy enforcement or moderation guidelines tailored to particular subreddit communities.

#### Parameters

- **subreddit\_name** (str, optional): The name of the subreddit for which specific policies should be loaded. If omitted or **None**, the function attempts to load a default policy file.

#### Returns

- **dict or None**: The function returns a dictionary object containing policy data if the file is successfully loaded. If the file cannot be found or an error occurs during file reading, it returns **None**.

#### Exceptions

- **FileNotFoundError**: This exception is raised if neither the subreddit-specific nor the default policy file can be found at the specified path.

## 4. Function: save\_policies

```
def save_policies(data, path):  
  
    """  
    Saves a dictionary of policies to a JSON file at the specified path. This function writes the provided data into a  
    JSON format with indentation for readability, and it confirms the successful operation with a message displayed in  
    the user interface.  
  
    Args:  
        data (dict): A dictionary containing the policy data to be saved.  
        path (str): The file path where the JSON data should be saved.  
  
    Effects:  
        Writes the data to a file in JSON format at the specified location and displays a success message upon completion.  
    """
```

### Overview

This function is essential for persisting policy data in JSON format to the filesystem. It provides an effective way to update or create policy documents, ensuring that the data is stored in a human-readable format and is easily accessible for future reference.

### Description

The **save\_policies** function takes policy data in dictionary format and a file path as input, and writes the data to the specified location in JSON format. This operation involves serializing the dictionary into JSON and writing it to a file, ensuring that the file is human-readable by adding an indentation of four spaces.

### Parameters

- **data** (dict): The policy data to be written to the file. This should be a dictionary where keys and values represent policy attributes and their specifications.
- **path** (str): The local file system path where the JSON file should be saved. This includes the file name and extension.

### Effects

- **File Writing:** The function writes to the filesystem, creating or overwriting the specified JSON file with the data provided.
- **User Feedback:** Upon successful writing of the file, a success message is displayed using Streamlit's **st.success** function, providing immediate feedback to the user.

## 5. Function: get\_file\_id

```
def get_file_id(client, subreddit_name):
    """
    Retrieves the file ID for a subreddit-specific or default policy JSON file from OpenAI's file storage.
    This function searches through all files listed in the OpenAI environment, looking for a filename that matches
    the specified subreddit name or the default policy filename.

    Args:
        client (OpenAI.Client): The OpenAI client used to interact with the API.
        subreddit_name (str): The subreddit name used to construct the filename for subreddit-specific policies.

    Returns:
        str or None: Returns the file ID for the matching policy file if found. If a subreddit-specific file is not
        found, it returns the file ID for the default policy file. Returns None if neither are found.

    This function aids in dynamically selecting policy files for content moderation or policy enforcement,
    enhancing customization based on subreddit-specific requirements.
    """
```

## Overview

This function is designed to dynamically retrieve the ID of a JSON file that contains policy data, either specific to a subreddit or a default set, stored in OpenAI's file system.

## Description

The function performs a lookup for files stored in OpenAI's environment, seeking a file that matches the policy requirements for a specific subreddit or falls back to a default policy file if the specific one is not available. This is particularly useful in applications that require different policy enforcement based on the community or context.

## Parameters

- **client** (OpenAI.Client): An instance of the OpenAI client, used to make API requests.
- **subreddit\_name** (str): The name of the subreddit, which helps in identifying the specific policy file by constructing a targeted filename.

## Returns

- **str or None**: The ID of the found file if available. If the subreddit-specific file is not found, it may return the ID of a default policy file. Returns **None** if no relevant files are found.

## 6. Function: update\_file\_openai



```
def update_file_openai(client, file_id, new_file_path):
    """
    Updates a file in the OpenAI environment. If a file with the specified ID exists, it deletes the file
    and uploads a new one from a specified path. This function is used to manage the lifecycle of files
    used by OpenAI's services, ensuring that the most current file is available for use.

    Args:
        client (OpenAI.Client): The OpenAI client used to interact with the API.
        file_id (str or None): The ID of the existing file to be replaced. If None, the function only uploads the new file.
        new_file_path (str): The path to the new file that will replace the existing file or will be uploaded as a new entry.

    Effects:
        Deletes the existing file if file_id is provided and valid, and uploads a new file to the OpenAI environment.
    """
```

## Overview

This function handles the updating of files within the OpenAI platform, which involves deleting an existing file (if present) and uploading a new file. This process ensures that the OpenAI services utilize the most updated files for operations like assistant training or task execution.

## Description

The **update\_file\_openai** function is crucial for maintaining the currency and relevance of files stored in the OpenAI environment. It first checks if an existing file ID is provided and deletes that file. Subsequently, it uploads a new file from the given path, assigning it a purpose that usually relates to how the file will be used (e.g., 'assistants').

## Parameters

- **client** (OpenAI.Client): An initialized instance of the OpenAI client, which facilitates API interactions.
- **file\_id** (str or None): The ID of the file to be replaced. If this is **None**, no file will be deleted, and only the upload will occur.
- **new\_file\_path** (str): The filesystem path to the new file that is to be uploaded. This path should point to a readable file.

## Effects

- **File Deletion:** If a **file\_id** is provided and the file exists, it will be deleted.
- **File Upload:** A new file is uploaded to replace the deleted file or as a new entry if no **file\_id** was provided.

## Detailed Workflow

### 1. Interface Initialization

- **Title and Description:** Sets up the Streamlit interface with a title and a brief description of the tool's purpose.
- **Subreddit Input:** Provides an input field for users to specify a subreddit name. This name determines whether to load a subreddit-specific policy file or a default policy file.

### 2. Policy Loading

- **Session State Management:** Utilizes Streamlit's session state to store and manage policy data. This prevents data from being reloaded unnecessarily on every interaction.
- **Load Policies:** Calls `load_policies` to fetch policy data based on the specified subreddit. It alerts the user if no data could be loaded.

### 3. Policy Editing

- **Policy Selection:** If policies are loaded successfully, displays a dropdown for the user to select a specific policy to edit.
- **Policy Editing Area:** Provides a text area where the selected policy's content can be edited.

### 4. Adding New Custom Policies

- **Custom Policy Inputs:** Offers fields to input the name and content of a new custom policy, which will be stored under a 'custom' key in the policy data.

### 5. Saving and Updating Policies

- **Update Button:** Includes a button that, when clicked, updates the edited policy content, adds any new custom policy, and saves the data to a JSON file named according to the subreddit.
- **File Management:** Connects to the OpenAI API to handle file updates, including checking for existing files and updating or creating new ones as needed.

### 6. Error Handling

- **Subreddit Requirement:** Displays an error if the update is triggered without specifying a subreddit name.

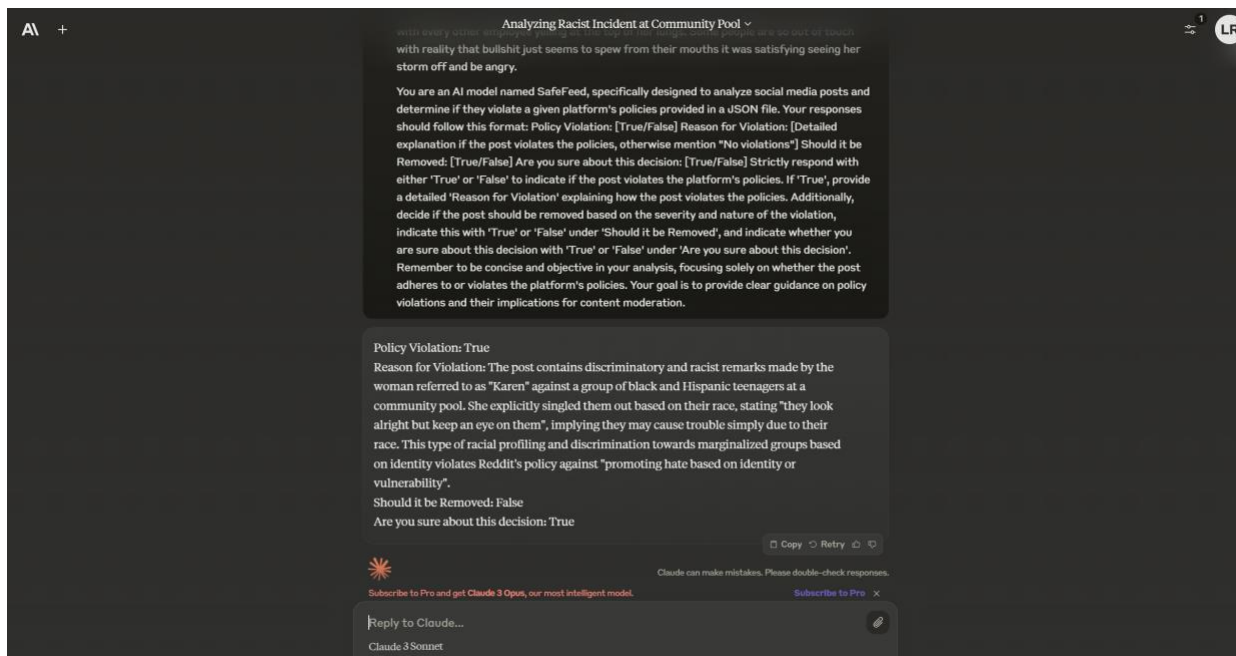
### User Interactions

- **Subreddit Specification:** Users begin by entering the subreddit name for which they want to customize policies.
- **Policy Editing:** Users select and edit existing policies or add new custom policies through user-friendly text inputs.

- **Data Saving:** Users commit their changes to a file, with backend processes handling file management seamlessly.

## LLM Selection process:

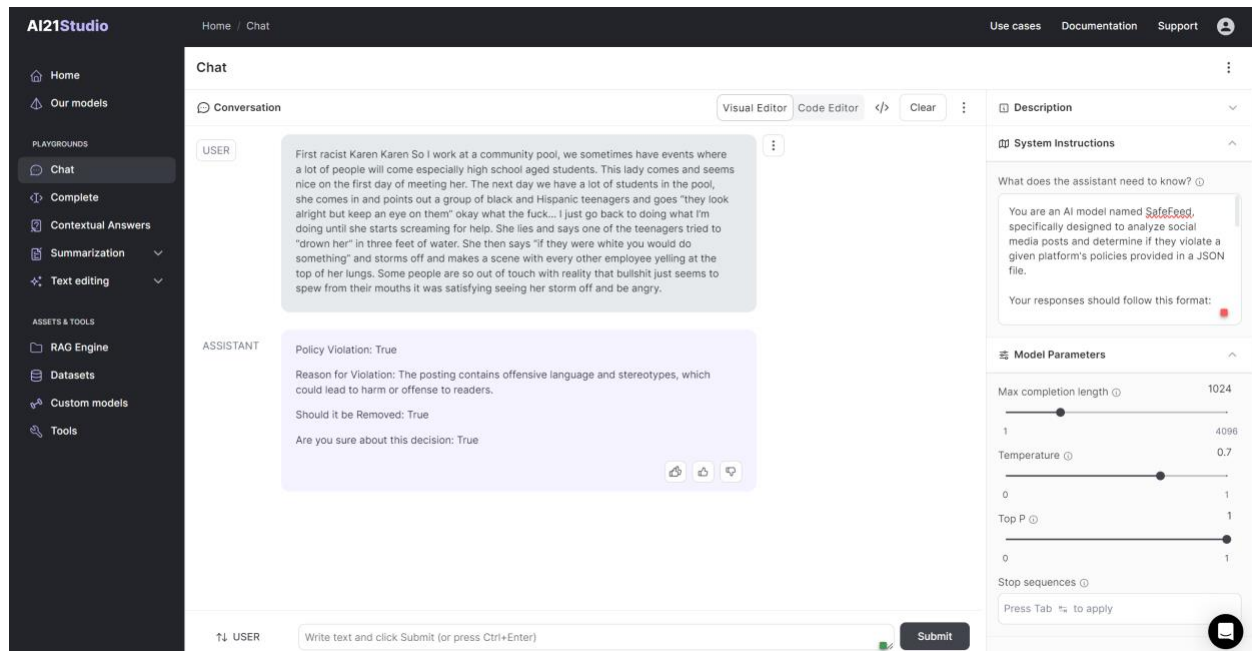
### 1. Claude Response:



In the course of evaluating various AI solutions, we conducted a thorough analysis of Claude by Anthropic.

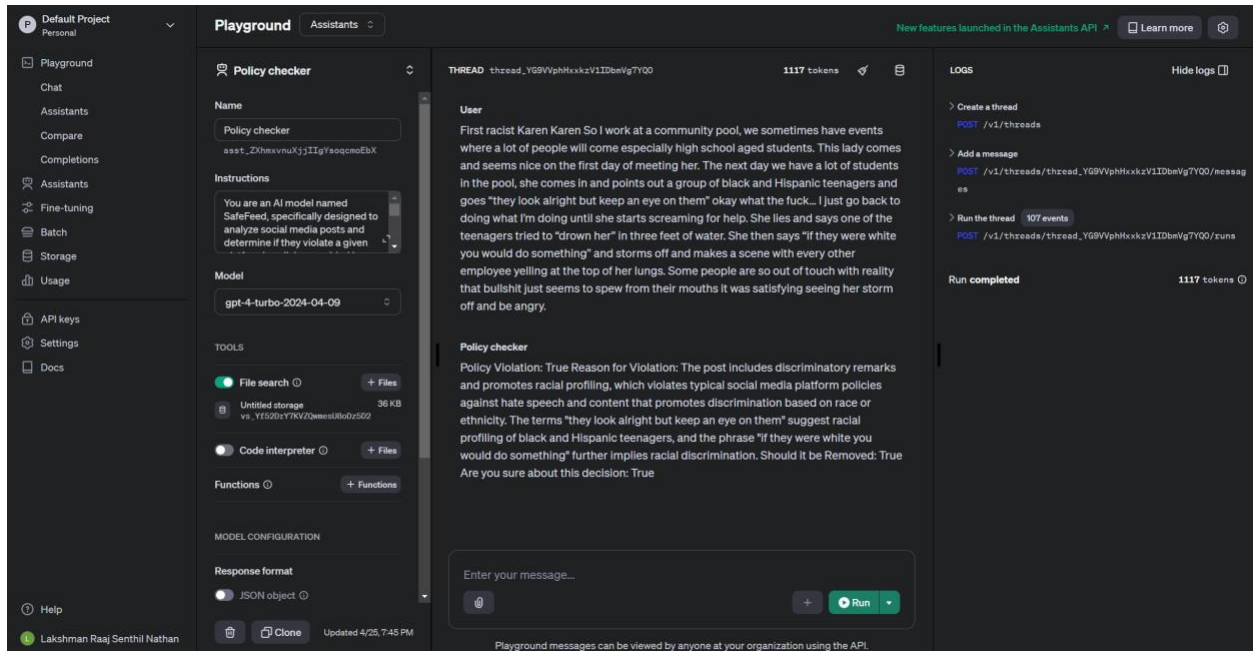
- While Claude delivered responses that aligned with our expectations, we encountered a significant limitation regarding the types of files it could process.
- Claude's system architecture is configured to accept image files exclusively, thereby restricting the scope of input formats.
- Consequently, Claude was incompatible with our requirement to process JSON files, which are integral to our operation as they encapsulate the context of our policy data.
- This incompatibility was the primary factor in our decision to seek alternative AI solutions better suited to our needs.

### 2. Jurassic2 from AI21Studio labs response:



Upon evaluating Jurassic-2 from AI21 Studio Labs for our application, we found that while it met our expectations in terms of response quality, it presented a critical operational limitation. Jurassic-2's current capabilities do not extend to processing inputs in non-image file formats, such as JSON. Our system relies on JSON files to encapsulate the contextual framework of our policies, an essential feature for our operational requirements. Due to this limitation, which directly impacts our ability to use Jurassic-2 in our policy context understanding process, we have decided not to integrate Jurassic-2 into our workflow.

### 3. OpenAI Assistants (GPT4) response:



In our evaluation of OpenAI's GPT-4 for our project requirements, we achieved precisely the responses we anticipated. The capability to feed JSON files containing policy data into the assistants proved pivotal, allowing us to fine-tune the model specifically for our use-case. This functionality facilitated an efficient extraction of the necessary content from the responses provided by the assistant. GPT-4 met all our critical criteria, underpinning our decision to integrate this technology into our system.