**Communication Analysis Tool: Project Overview and Architecture**

This document provides a high-level overview of the Communication Analysis CLI tool's architecture and the interaction between its core Python modules. The tool is designed with a modular approach, where each component handles a specific aspect of the communication analysis pipeline.

**1. Core Modules**

The application is structured around several key Python modules, each residing in the modules/ directory:

* **main.py**:
  + **Role:** The entry point of the application. It provides the command-line interface (CLI) and orchestrates user interaction.
  + **Functionality:** Displays the main menu, handles user input for selecting analysis options, and calls the appropriate functions from report\_generator.py based on user choice. It also manages the flow for custom message input, including prompting for message type, sender, and conversation ID.
* **spam\_detector.py**:
  + **Role:** Identifies messages as spam or legitimate (ham).
  + **Functionality:** Likely implements a Bayesian filtering approach. It's initialized with training data to learn patterns associated with spam and ham messages. Its predict() method takes a message string and returns a boolean (True for spam, False for ham).
* **sentiment\_analyzer.py**:
  + **Role:** Determines the emotional tone of a message.
  + **Functionality:** Uses a rule-based approach with predefined lexicons (positive\_words, negative\_words). It counts the occurrences of words from these lexicons in a given text to classify its sentiment as 'positive', 'negative', or 'neutral'.
* **style\_analyzer.py**:
  + **Role:** Assesses the writing style and formality of a message.
  + **Functionality:** Calculates a numerical style score (e.g., based on sentence length, word complexity, or specific formal/informal phrases) and categorizes the message's formality (e.g., 'formal', 'informal').
* **metrics\_calculator.py**:
  + **Role:** Computes aggregate metrics and behavioral insights from a list of analyzed messages.
  + **Functionality:** Takes the structured output from report\_generator (a list of message dictionaries) and calculates:
    - Spam distribution (count of SPAM vs. HAM).
    - Sentiment breakdown (count of positive, neutral, negative).
    - Average style score.
    - Formality breakdown.
    - Top senders (frequency of messages per sender).
    - Average response times within conversations (by grouping messages by Conversation ID and analyzing timestamps).
    - Generates behavioral suggestions based on these metrics.
* **report\_generator.py**:
  + **Role:** The central processing unit that orchestrates the analysis of input data and compiles the final reports.
  + **Functionality:**
    - **Input Handling:** Reads raw text files from data/sample\_emails/ or processes direct user input.
    - **Format Detection:** Employs helper functions (\_is\_whatsapp\_format, \_has\_email\_boundary) to intelligently detect the format of input files (single email, multi-email, WhatsApp chat).
    - **Parsing:** Uses specialized parsing functions (\_parse\_single\_email\_block, \_parse\_multi\_email\_file, \_parse\_whatsapp\_content) to extract message content and metadata (Sender, Conversation ID, Timestamp, Subject) from various formats.
    - **Analysis Orchestration:** For each extracted message, it calls the predict()/analyze() methods of spam\_detector, sentiment\_analyzer, and style\_analyzer.
    - **Result Compilation:** Gathers all analysis results and extracted metadata into a list of structured dictionaries.
    - **Metric Calculation:** Passes the compiled results to metrics\_calculator.py to obtain summary and behavioral data.
    - **Report Generation:** Writes a comprehensive human-readable report to a .txt file in data/reports/.
    - **CLI Feedback:** Prints a condensed summary of metrics and behavioral insights directly to the console using \_print\_summary\_to\_cli.

**2. Data Flow and Interactions**

The analysis process follows a clear pipeline:

1. **Input Acquisition:** main.py prompts the user to choose an input method (full dataset, custom file, or typed message).
2. **Raw Data Reading & Parsing:** report\_generator.py reads the raw text content of selected files or directly receives typed input. It then intelligently parses this raw content based on detected format:
   * **Emails:** Headers (Sender, Date, Convo ID, Subject) and message body are extracted.
   * **WhatsApp:** Each line is parsed to extract sender, timestamp, and message. Multi-line messages are reassembled.
   * **Multi-Email Files:** The file is split by ---EMAIL\_BOUNDARY---, and each block is parsed as a separate email.
3. **Individual Message Analysis:** For each distinct message (whether from a file or typed), report\_generator.py sends its content to:
   * spam\_detector.predict()
   * sentiment\_analyzer.analyze()
   * style\_analyzer.analyze() The results (spam status, sentiment, style score, formality) are then added to the message's dictionary along with its extracted metadata.
4. **Aggregate Metric Calculation:** Once all messages from the input source are processed and compiled into a results list, this list is passed to metrics\_calculator.py.
5. **Report Generation & Feedback:** report\_generator.py receives the summary and behavioral insights from metrics\_calculator.py. It then formats and writes these insights, along with the individual message analyses, into a detailed report file. Finally, it calls \_print\_summary\_to\_cli to display a quick overview to the user in the terminal.

**3. Error Handling**

The modules incorporate try-except blocks to gracefully handle common issues such as:

* File not found errors.
* ValueError during timestamp parsing (e.g., if a date string is in an unexpected format).
* General exceptions during message processing to ensure the application doesn't crash but logs the error within the report.

This modular design promotes reusability, maintainability, and allows for easier extension of new analysis features or input formats in the future.