MAILA: An AI-Powered Implementation of Guerrilla Mail

CHRISTOPHER E. CORTEZ, University of Nottingham, United Kingdom

1 Introduction

Maila[2] is a chatbot that provides a conversational interface for the Guerrilla Mail API[1], a temporary email service. Users can conversationally start, restore, and manage a disposable email inbox, with options to check, view, download, or delete email(s).

Maila also supports non-transactional chatbot interactions, including *Small Talk* (handling greetings and pleasantries), *Question and Answering* (providing answers from a dataset), *Personalization* (learning and using the user's name), and *Discoverability* (built-in help system to explain its own capabilities).

These features are enabled by two key internal systems: *Intent Classification*, which routes the user's query to the correct module, and *Context Tracking*, which manages Maila's conversational state to handle multi-step tasks.

2 Core Architecture and Design

2.1 System Design

Figure 1 shows the simple file tree of Maila:

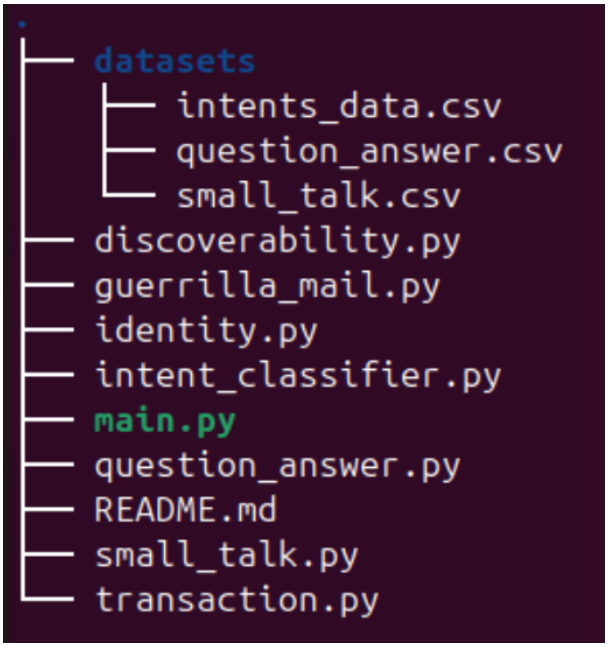

Fig. 1. Maila's file tree

Maila is a modular system built on the principle of separation of concerns, improving the maintainability of the code and allowing for features to be added or modified easily. The `main.py` file is the centralizing point, managing

Author's Contact Information: Christopher E. Cortez, pscxx18@nottingham.ac.uk, University of Nottingham, Nottingham, United Kingdom.

Maila's core logic and GUI. It receives user queries, directs them to the appropriate handler, and manages key variables such as `chat_stack` (used for context tracking), `username`, and the disposable email address.

2.2 Intent Classification

The `intent_classifier.py` module functions as the system's primary routing mechanism. It operates on a dataset mapping text phrases to a primary intent and an optional subintent. Upon initialization, the module builds a TF-IDF[1] model from this dataset. When a user query is received, it is vectorized, and cosine similarity is employed to identify the best-matching phrase in the model. The resulting intent and subintent are returned to `main.py`, which routes the original query to the appropriate handler (e.g., `discoverability.py`) for processing.

Subintents provide a granular classification; they specify *how* a handler should process a query, allowing Maila to bypass standard dialogue flows for more specific user requests.

2.3 Context Tracking

To manage multi-step interactions, Maila relies on a stack (`chat_stack`) for context tracking, functioning as a LIFO stack where `chat_stack[-1]` represents Maila's current state. The stack is initialized with the `'normal'` state.

There are fourteen states in total, split into three task groups (*identity*, *discoverability*, and *email*). States represent the natural progression of a task or conversation, and are continually added to `chat_stack` as triggered. When the particular conversation (or transaction) is completed, all states of that particular group are removed from `chat_stack`.

States are tightly integrated with intents, enabling priority-based handling; states and intents placed earlier in `main.py`'s `get_bot_response` method are evaluated first, allowing nested task flows as shown in this example:

(1) A general "*help*" query pushes a new state onto `chat_stack`: `['normal','general_help_loop']`. Maila is now in this state.
(2) If the user then asks about *capabilities*, the stack becomes: `['normal','general_help_loop', 'capabilities_help']`. When this sub-task is complete, both *discoverability* states are popped, returning the stack to `['normal']`.
(3) Crucially, certain tasks are prioritized. If the user says "*change my name*" (an *Identity* task) while in a *discoverability* state, the 'awaiting_name' state is pushed on top: `[...'capabilities_help','awaiting_name']`. Once the identity task is complete, that state is popped, and the *discoverability* task is resumed.

2.4 Query Flow

All user input is processed by the `get_bot_response` method within `main.py`, which follows a precise order of operations to handle said query:

(1) **Input**: The user enters and submits a query via the GUI.
(2) **Universal Command Check**: The query is checked against the five universal commands of the chatbot; if there's a match, the command logic is directly carried out.
(3) **Intent Determination**: The query is sent to the intent classifier, where it determines the user's intent (and subintent) based on its trained model.
(4) **Handler Routing**: Based on the returned intent or the current state of the bot, the query is routed toward a particular handler, which processes it accordingly.
(5) **Handler Response**: The handler returns a response, as well as variables and an updated state depending on which handler processed the query; if the updated state is `'normal'`, then Maila will assume the task is complete and pop all states of that particular group from the stack.

[1]Term Frequency - Inverse Document Frequency

(6) **Display Response**: Lastly, the `response_text` returned by the handler will be displayed to the user on the GUI.

2.5 Small Talk & Question and Answering

Functionally, `question_answer.py` and `small_talk.py` are dialogue and information retrieval systems, structurally similar to `intent_classifier.py`. Both modules process user queries by leveraging a TF-IDF model trained on their respective datasets. A query is vectorized and compared against the trained data using cosine similarity to find the best-matching response. While architecturally similar, their text processing and response logic are slightly distinguished.

2.5.1 ***Question Answering.*** To ensure high-precision retrieval, this module implements specific preprocessing steps:

- **Stopword Removal**: Stopwords are removed during the vectorization process to focus on the core semantic meaning of the user's query.
- **High-Confidence Thresehold**: A high similarity threshold is enforced to prevent false positive matches.
- **Fallback Responses**: If a query fails to meet this threshold (i.e., the answer is unknown), the system delivers a simple, effective fallback response.

2.5.2 ***Small Talk.*** This module is optimized for natural, conversational interaction:

- **Stopword Retention**: Stopwords are intentionally retained during vectorization, as they are essential to the grammatical structure and meaning of conversational phrases (e.g., "*How are you?*").
- **Response Randomization**: The `small_talk.csv` dataset provides multiple response options for each input phrase. When a match occurs, a response is selected at random to make the dialogue feel more dynamic.
- **Personalization**: Responses can incorporate the `username` variable (if one has been declared) to personalize the conversation.

2.6 Identity

`Identity.py` relies on the provided subintent to properly handle a user's identity request. There are four subintents associated with identity:

- **Identification**: Responds to "*Who am I?*" and similar queries. If the name is unknown, it triggers a confirmation state (`'awaiting_name_confirm'`).
- **NameChange**: Triggers the `'awaiting_name'` state to prompt the user for their name.
- **NameDirect**: Extracts the name directly from the query (e.g., "*My name is Chris*") using tokenization and stopword removal.
- **NameDelete**: Forgets the stored `username` variable.

2.7 Discoverability

The `Discoverability.py` module implements Maila's help functionality. Its operation is managed through two primary states: `'general_help_loop'` and `'capabilities_help'`.

When a user makes an unspecified help query, the `'general_help_loop'` state is pushed onto `chat_stack`. The chatbot then prompts the user, offering information on its general *capabilities*, *identification*, or *commands*. If the user indicates *capabilities*, the `'capabilities_help'` state is added to `chat_stack`. Maila then offers to provide a detailed explanation of one of its core functions: small talk, question and answer, identity management, or email services. Once the user's selection is addressed, the relevant discoverability states are removed from `chat_stack`, concluding the help interaction.

`Discoverability.py` also supports subintent routing, which allows users to bypass the standard dialogue flow. For example, a query such as “Tell me about small talk” will directly trigger the appropriate response, regardless of whether the user is already in a discoverability state.

2.8 Transaction

The `Transaction.py` and `GuerrillaMail.py` modules collectively manage the conversational implementation of the Guerrilla Mail API client. `GuerrillaMail.py` contains the core methods for API communication, while `Transaction.py` processes user queries. `Transaction.py` determines the required action based on the current conversational state or the provided subintent, invoking methods from `GuerrillaMail.py` as needed.

Figure 2, shows the transaction flow:

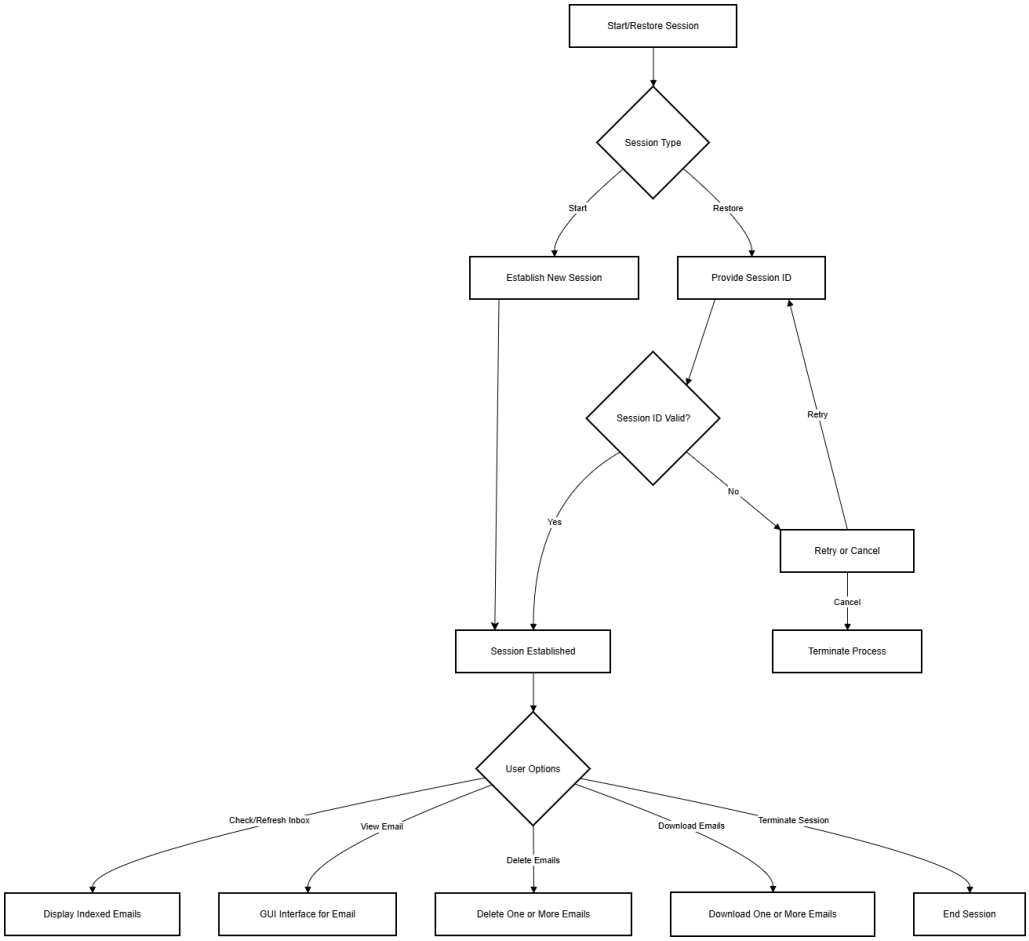

Fig. 2. Maila’s transaction flow

The initial action is session establishment; a user can either start a new session or restore a previous one by providing a valid session ID. If a user attempts to manage emails without an active session, Maila will prompt

them to start one. Once a session is established, Maila provides the user with a disposable email address and the corresponding session ID. The user then has access to several management options:

- **Check Inbox**: The user asks Maila to check the inbox, and it returns an indexed list of emails received by the address.
- **View**: Based on the indexed list, the user can ask to open/view a particular email, it will be displayed in a separate GUI interface.
- **Delete**: The user can request Maila to delete one or more emails from the inbox; if one asks to delete all, Maila will prompt for confirmation.
- **Download**: The user can request Maila to download one or more emails into a local file; they are downloaded as HTML and appear in a generated `downloads/{sessionID}` folder.
- **End Session**: The user can ask Maila to end the current session, wiping the email address and session ID locally.

Users are not required to end the session manually to perform other tasks. It is possible to restore a session using its session ID; however, the Guerrilla Mail server automatically terminates sessions after 18 minutes of inactivity and deletes all emails after one hour. For management tasks (view, delete, download), if the user's request does not specify which email(s) to act upon, Maila will prompt for clarification before proceeding.

3 Conversational Design

3.1 GUI Design

Figure 3 shows the GUI with some sample dialogue:

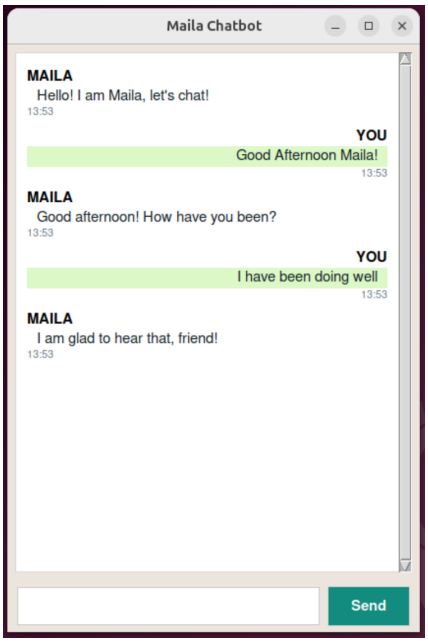

Fig. 3. Maila GUI with sample dialogue

The GUI was designed to provide a familiar interface, mimicking a typical messaging app. Maila's responses are on the left, and the user's responses are on the right (highlighted in green), along with timestamps below

each message. If a user tells Maila their name, 'YOU' is replaced with the provided name. The interface is easily resizable via expansion on either side or by interacting with the buttons on the top right; clicking the 'X' button terminates the application entirely. The input interface is on the bottom, and users can submit their messages via the provided '*send*' button or by pressing '*enter*' on their keyboard.

3.2 Discoverability

See Section 2.7 for a detailed description of discoverability; it provides help and discoverability through conversational dialogue, allowing users to learn about Maila's various capabilities via conversation or direct querying.

When Maila enters a new state, the dialogue reflects the state, providing pointers to the user on how to proceed. For example, when one enters the help loop, a possible initialization dialogue is, "*I can help. Are you interested in my commands, identification, or my capabilities?*", informing the user on what they can inquire further on and trying to get them to reply in a particular way; alternatively, the command *"what now"* can provide the user direction.

3.2.1 ***Commands.*** Either through branching dialogue or direct inquiry, commands can be discovered and utilized by the user; as shown in Section 2.4, they take ultimate priority. Designed to assist the user in case they get lost, confused, or trigger something in error, there are five universal commands:

- **WHERE AM I**: Returns the current state from `chat_stack`.
- **WHAT NOW**: Based on the current state, it informs the user of the next course of action.
- **GO BACK**: Goes back one state in `chat_stack` and returns the new state.
- **CANCEL**: Cancels the ongoing task in its entirety (removes the entire group from `chat_stack`, similar to if the task was completed).
- **REPEAT**: Repeats Maila's initial dialogue for the current state.

3.3 Personalization

Maila can remember and use one's name, the process of which is described in Section 2.6. Additionally, Maila incorporates the name into small talk dialogue and the GUI, as described in Section 2.5.2 and 3.1.

3.4 Context Tracking

Maila keeps track of context as described in Section 2.3. The design of `chat_stack` and the states that it utilizes allows a user to start a new task/conversion nested in an ongoing task, ask for help, engage in small talk, etc., without losing their place in the ongoing transaction.

3.5 Confirmations

Confirmations are used sparingly; generally, if a user queries the right way with the correct information, the action is carried out. However, explicit[2] confirmations are utilized in two[3] primary scenarios: confirming the initiation of a new task, and preventing destructive or irreversible actions. Each time a confirmation is triggered, it adds a new state to `chat_stack`.

For the former, if a user queries Maila, "*What is my name?*", without having a name set, or queries, "*check emails*", without having an active session, Maila will inform the user of this and ask if they would like to start a session or set their name, awaiting a yes or no answer.

For the latter, if the user explicitly asks Maila to delete all of their emails from the inbox or to end their current email session, it prompts the user for confirmation.

[2]Implicit confirmations are used in `identity.py` and `transaction.py`
[3]*Discoverability* loops subtly utilize them

3.6 Error Handling

Critical errors, such as the failure of the intent classifier to load and train data, are immediately outputted in both the console and the GUI. If Maila fails to connect to the API, it responds with this information and suggests that the user checks their internet connection. All technical errors follow this same pattern.

Maila reprompts the users in certain states, such as in the discoverability loop when it receives an unexpected query. The commands provided also allow the user to break out of loops, tasks/conversations themselves.

If Maila receives a query that fails to be classified (either by the intent classifier or by the handler), it responds with a fallback message, such as, *"Forgive me, but I'm unable to recognize what you are saying."* or *"I'm afraid I don't have the answer to that."*.

3.7 Prompt Design

Prompt design is implemented to ensure clear user guidance. As described in Section 2.3, Maila's query expectations are tied to the current state in `chat_stack`. When Maila enters a new state, the resulting prompt is a confirmation, direction, or a specific instruction, such as, *"Which email index would you like to view? Please enter a number."*

To enhance the conversational experience and avoid repetition, Maila employs response randomization. As seen in the *transaction*, *identity*, *small talk*[4], and *discoverability* modules, responses are randomly selected from a predefined list of templates. This ensures that common interactions, such as starting a session or providing help, feel more dynamic and less robotic.

4 Evaluation

4.1 Performance Evaluation

We chose to evaluate the intent classifier due to the critical role it plays in query routing (and the overall system); to measure performance, a script was written that does the following:

(1) **Preprocessing**: It replicates the exact preprocessing functions found in `intent_classifier.py`
(2) **Train/Test Split**: It takes the `intents_data.csv` and splits it into 70-30 training-testing set.
(3) **Classification**: It trains the TF-IDF model on the training data; for each test query, it finds the most similar phrase in the testing data and copies its intent as the prediction.
(4) **Report**: It returns the overall accuracy, classification report (Precision, Recall, F1-Score), and a confusion matrix.

Please see figure 4 for the classification report and figure 5 for the confusion matrix.

4.2 Usability Testing

Due to time constraints, the feasibility of conducting proper user testing is limited. Instead, an LLM (Gemini) was given a high-level overview of the system and was instructed to generate 10 complex user flows, providing user input and the action/response that should happen. These flows were then manually run through the system. Descriptions of each of the flows are as follows:

(1) Tests the email transaction in a simple start-to-finish manner.
(2) Tests context switching by measuring Maila's ability to handle nested tasks and interruptions.
(3) Tests clarification prompts, error handling, and commands.
(4) Tests handling queries without a session and session restoration.
(5) Tests the identity flow from an unknown state and the 'repeat' command.
(6) Tests Maila's ability to handle Q&A and small talk during an active email session.

[4]The varied responses are built directly into `small_talk.csv`, but the concept remains the same

(7) Tests Maila's error handling for invalid inputs and actions on nonexistent data.
(8) Tests the integrity of `chat_stack` with multiple nested interruptions.
(9) Tests the classifier's ability to handle synonyms and the confirmation state's strictness.
(10) Tests vague user replies and non-standard exit phrases.

```
--- PERFORMANCE RESULTS ---
Accuracy: 81.60%
--------------------------
Classification Report:
                    precision    recall  f1-score   support

   Discoverability       0.89      0.76      0.82        21
             Email       0.96      0.88      0.92        26
IdentityManagement       0.89      0.85      0.87        20
 QuestionAnswering       0.58      0.79      0.67        19
         SmallTalk       0.82      0.79      0.81        39

          accuracy                           0.82       125
         macro avg       0.83      0.82      0.82       125
      weighted avg       0.83      0.82      0.82       125
```

Fig. 4. Classification Report of the intent classifier's evaluation script

5 Discussion

5.1 Performance Evaluation Results

The classifier achieved an overall accuracy of 81.6%. The detailed F1-scores are shown in the classification report (Figure 4), and a heatmap of the results is presented in the confusion matrix (Figure 5).

The strongest component is the `Email` intent, which achieved an F1-score of 0.92. This was supported by high precision (0.96) and recall (0.88), indicating that Maila is both confident and correct when predicting a user's email-related tasks. The `IdentityManagement` intent also performed very well, with an F1-score of 0.87.

The main weakness is the `QuestionAnswering` intent, which had the lowest F1-score of 0.67; revealing an imbalance between the recall (0.79) and the precision (0.58), suggesting that Maila finds the real questions, but mislabels many other intents as `QuestionAnswering`.

The confusion matrix readily confirms this, of the 19 test queries with a true `QuestionAnswering` intent, 15 were correctly classified. However, 11 queries from other intents were incorrectly predicted as `QuestionAnswering`. This contrasts[5] with the `Email` intent, which only "stole" 1 query.

5.2 Test Flow Results

A usability test was run using the 10 test flows (as described in 4.2). Six flows (1, 2, 4, 5, 6, 8) were completed successfully. The remaining flows revealed two distinct issues:

[5]Of the 23 total mistakes, 18 of them were misclassifications of intents to `SmallTalk` or `QuestionAnswering`, while only 5 intents were misclassified as `Discoverability`, `Email`, and `IdentityManagement`

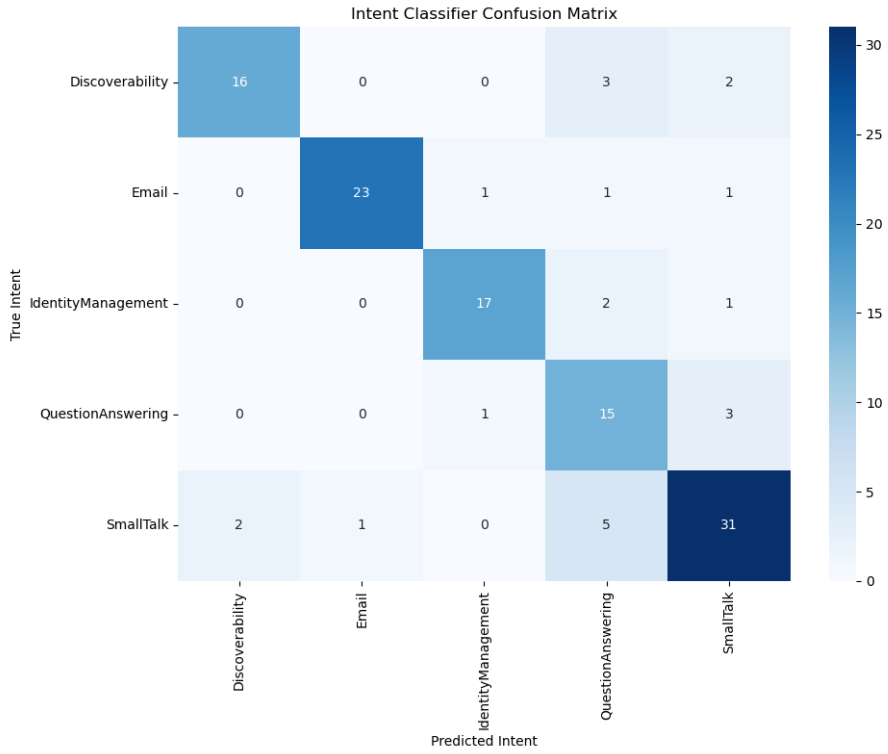

Fig. 5. Confusion Matrix of the intent classifier's evaluation script

(1) **Logic Flaw (Flows 3 & 7)**: Maila misinterpreted input while in an email management state.
(2) **Robustness (Flows 9 & 10)**: These flows, which tested vague or alternate user phrasing, had a low success rate of only 47%.

5.3 Results Discussion

The evaluation shows that the `intents_data.csv` file lacks sufficient data necessary to handle various non-standard inputs. This is the main cause for both the poor 47% success rate on alternate phrasing (Flows 9-10) and the high confusion between `QuestionAnswer` and `SmallTalk` seen in the confusion matrix (Figure 5). A hilarious example is seen in figure 6, when a user's query, "*I want to trash everything*" (to delete all emails), is misclassified as `SmallTalk`, to which Maila replied, "*How is life treating you, friend?*". This is a failure in transparency, as a user would have no way of understanding why their intent was classified as small talk and why it failed to carry out the desired action.

Furthermore, flow 3 and 7 revealed a critical logic flaw in the email-awaiting states. These states route inputs into `transaction.py` blocks designed to extract only numbers, ignoring all other text. This is potentially destructive, as a user in the 'awaiting_delete_index' state who inputs "*view email 1*" (either because they changed their mind or lost track of their place) will have Maila delete email 1, as that is the index of the email that was extracted.

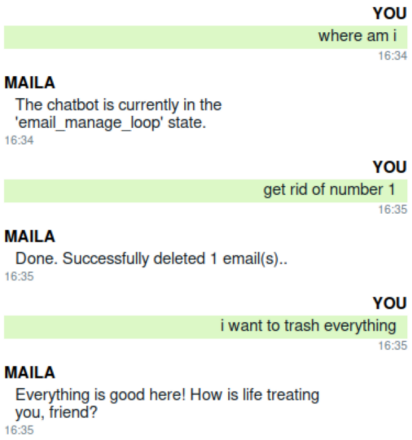

Fig. 6. Alternate phrasing being misclassified in flow 9

This flaw violates[6] transparency and accountability; while the user did trigger Maila's delete state, the process to switch actions, cancel, go back is not necessarily clear. Regardless of the current state, their `view` intent was completely ignored, which caused Maila to delete the exact email they wanted to look at. A more accountable system might implement guardrails for all deletion actions (not just for '*delete all*' in the current implementation) and a more transparent system would inform the user that they can '*cancel*' or '*go back*' upon initialization.

5.4 Application & Ethics Reflection

The core function of Maila is to provide conversational access to the Guerrilla Mail API, granting them access to a temporary email address. Realistically, a user who may struggle[7] with the web interface of Guerrilla Mail (or similar services) may find genuine use in Maila's conversational implementation and interface.

The nature of the service makes Maila inherently pro-privacy, empowering users by allowing them to shield their primary email from potential spam, data harvesting, and unwanted tracking. Guerrilla Mail's own servers end sessions after 18 minutes and delete emails after an hour, making long-term storage and security a non-issue, assuming Guerrilla Mail is trustworthy themselves[8].

Both environmental and social sustainability apply to Maila. Modern AIs such as LLMs tend to have significant environmental footprints due to data and computational requirements. Maila utilizes TF-IDF vectorization and cosine similarity, which are computationally lightweight and require far less energy compared to newer methods and models.

Maila's social value is twofold; it can contribute positively to individual well-being by reducing spam and protecting user identity. Conversely, it can be used for socially unsustainable activities such as creating fake accounts or exploiting services. A specific persona could be developed that discourages malicious use, but the ultimate decision lies with the user.

[6]One can argue that it violates safety via unintended data loss
[7]The elderly, technologically illiterate, etc.
[8]Consider the transparency issue here

References

[1] 2006. Guerrilla Mail. https://www.guerrillamail.com/
[2] Christopher Cortez. 2025. *MAILA: An AI-Powered Implementation of Guerrilla Mail.* https://github.com/cec5/MAILA