

Tze Zhe Brandon Lim, Mikko Tripakis, Nicholas Mamisashvili

Agenda AI: LLM-Powered Scheduling Assistant for Emails

1. Introduction

Agenda AI helps professionals and academics manage the chaos of scheduling hidden in messy email inboxes. Many meeting-related emails are informal, missing calendar invites, or buried in long threads, making it easy to miss important details. Our goal is to build an assistant that can understand these emails, extract relevant meeting information, and answer natural language questions about scheduling.

To do this, we combine a Retrieval-Augmented Generation (RAG) system with an agent-based reasoning loop. Emails are pre-processed with metadata and enhanced with summaries and inferred questions using GPT-4o mini. These enriched documents are embedded using sentence-transformers and stored in a vector database. A retriever surfaces relevant documents, which are re-ranked before being passed to the agent.

We evaluate Agenda AI using a small benchmark of manually constructed user queries. These test the system’s ability to (1) find events on specific dates, (2) handle vague time references like “next week,” and (3) resist hallucination when presented with false premises. We measure success using an LLM-based evaluator that checks whether the response is semantically close to the correct answer.

1.1 Example Task

The user requests information about their schedule for a particular date, and follows up with a request for additional information. The model fetches the relevant emails to answer the initial question and the follow-up question.

User: What meetings do I have on June 5?

Agent:

Here’s your schedule for June 5:

1. Meeting: GE End World Hunger Contract — 9:00–10:30 AM, Conf. Room EB38C2
2. Presentation: MDEA Delphi App — 4:00–5:00 PM, Room 3125B
3. Happy Hour — 5:00–8:00 PM, Front Porch Pub

User: Add my flight info for june and show me the full schedule

Agent:

Your flight information has been added to your calendar. Here's your full schedule for June 5:

1. Meeting: GE End World Hunger Contract — 9:00–10:30 AM, Conf. Room EB38C2
2. Presentation: MDEA Delphi App — 4:00–5:00 PM, Room 3125B
3. Happy Hour — 5:00–8:00 PM, Front Porch Pub

Additionally, you have the following flights scheduled:

- Flight to Memphis: Departs June 4 at 7:45 AM, arrives 9:30 AM (Seat 11C)
- Return Flight: Departs June 4 at 5:55 PM, arrives 7:40 PM (Seat 12C)

2. Methods

2.1 Dataset and Preprocessing

Our goal is to provide the agent with a realistic and challenging set of emails for evaluation. We begin with the Enron email corpus, a publicly available dataset on [Kaggle](#) that contains over 500,000 emails from multiple users. We randomly sample 200,000 emails and identify the four most active users based on email volume.

For each of these users, we aim to curate a set of 3,000 high-quality emails. The processing pipeline, implemented in [process_emails.py](#), involves several stages. First, each email is parsed using Python's email library to extract metadata such as sender, recipient, subject, date, and plain text content. We then apply filters to exclude sent emails, newsletters, and transcripts, and remove duplicates.

To identify scheduling-related content, we use a simple rule-based classifier that checks for the presence of meeting-related keywords in the subject or body of the email. If more than 3,000 emails are available for a user, we retain all meeting-related emails and randomly sample from the rest to maintain balance.

In a realistic scheduling scenario, a person's schedule for the upcoming week is often shaped by emails received in the previous one to two weeks. To reflect this, we use the four user datasets generated in [process_emails.py](#) and analyze them in [data_cleaning.ipynb](#). Through this analysis, we identify Kay Mann's inbox as having the highest density of scheduling-related content as well as a good balance of personal and professional emails. From her data, we select a focused subset of 154 emails from the end of May to the beginning of June 2001 for evaluation.

2.2 Retrieval Architecture

To support natural language queries over an unstructured inbox, we use a Retrieval-Augmented Generation (RAG) system. The indexing pipeline is implemented in [index_emails.py](#). Each email is first converted into a document that includes both the raw email content and metadata such as sender, recipient, subject line, and sent date. This metadata is preserved during indexing and later used to ground the agent's responses with useful context.

To further improve the usefulness of retrieval, each document is enhanced with additional LLM-generated information. Specifically, we prompt GPT-4o mini to extract key topics (such as specific dates, times, locations, and participants) as well as potential questions the email could answer (see Figure 1). These enhancements are appended to the original email text and stored alongside it in the vector store, which emails to a 768-dimensional vector space. We use Chroma for the vector store, with [sentence-transformers/all-mpnet-base-v2](#) as the embeddings model.

```
Please analyze the following email text and provide:

1. 3-4 main topics covered in the email, including dates, times, locations, people, and entities mentioned.
2. 3-4 questions this email could answer, especially related to scheduling (i.e., "When is the meeting?")

Format your response with clear headings for each section. Be concise and to the point.

EMAIL:
{text}
```

Figure 1: The prompt used to add additional context to emails for aiding retrieval.

The retrieval tool first runs a similarity search to fetch the 25 most relevant documents based on cosine similarity. We then re-rank those documents using [BAAI/bge-reranker-large](#), and return the top five most relevant documents as context. Adding LLM-generated context to our documents as well as implementing additional re-ranking proved to be especially useful for retrieval. Without any task-specific fine-tuning, the system can identify relevant messages and answer complex scheduling queries grounded in real email content.

2.3 Agent Architecture

The agent architecture follows two modes: a graph-based router and a ReAct-style agent, both built using the LangChain Python library. Figure 2 compares the two modes.

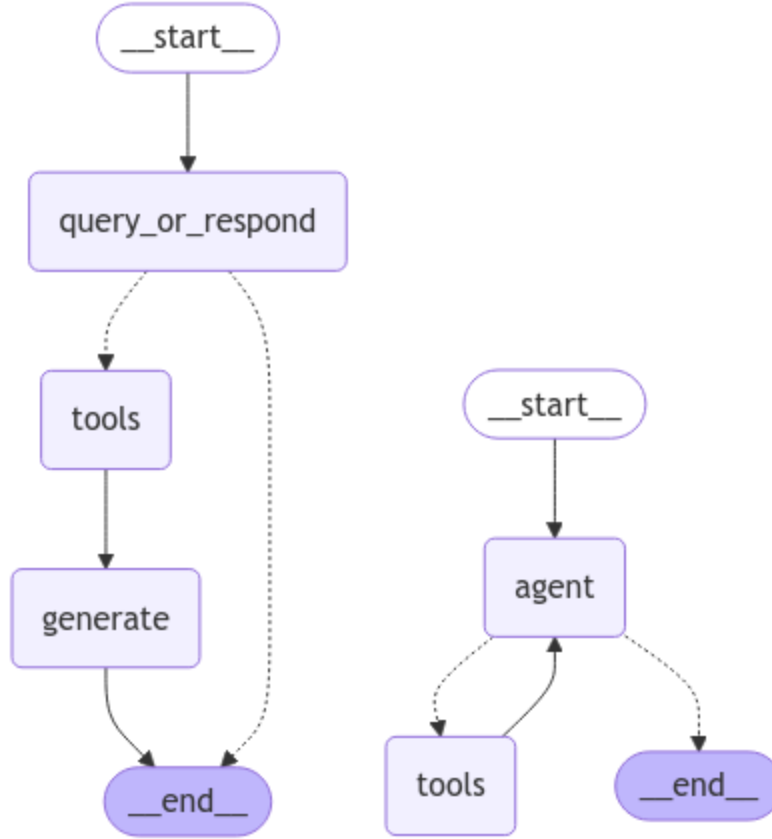


Figure 2: Diagram of the two agent architectures used in our system. The left shows the graph-based agent with separate routing, tool invocation, and response generation nodes. The right shows the ReAct agent, which dynamically decides when to use tools and when to respond.

In the graph-based mode, implemented in [llama.py](#), we use [Llama-3.1-8B-Instruct](#) to drive both decision-making and generation. When a user submits a query, it enters the graph through the `query_or_respond` node. This step determines whether the query is related to meetings, events, or scheduling using a classification prompt. If the model predicts that the query is scheduling-related, we force the model to invoke the `retrieve` tool described in section 2.2 (line 40). This step was necessary due to limitations in this smaller version of Llama: we observed that it struggled to decide to invoke the retrieval tool and frequently produced hallucinated responses when left to decide autonomously. Once the top documents are selected, the flow continues to the `generate` node (line 84). This node constructs a system prompt that combines the retrieved email content with associated metadata (like subject and date), and asks the model to generate a concise response.

In the ReAct-style agent, implemented in [4o_mini.py](#), we use GPT-4o mini as the underlying model, due to Llama’s difficulty in autonomously deciding when to use tools. This agent operates using the standard Thought → Action → Observation loop, enabling it to reason step-by-step through a query. Unlike the graph-based approach, the ReAct agent dynamically

decides whether to respond directly or invoke one of the available tools, and is able to make multiple tool calls before coming back to the user. In our evaluation, we focused on the `retrieve` tool, but we also implemented a prototype of an `add_to_calendar` tool which integrates with Google Calendar and allows the agent to add events to a user's calendar. The system is initialized with a concise system prompt (line 82) that instructs the agent to focus on scheduling-related questions. This setup allows the agent to handle a wider range of queries while staying focused on responding to scheduling-related content based on retrieved emails.

3. Evaluation

3.1 Testing Attempts

Testing our scheduling agent presented unique challenges unlike traditional software validation, as conversational AI outputs cannot be evaluated through simple correctness checks. The complexity stemmed from the diverse formats in which scheduling information could be effectively presented, ranging from structured bullet points to differentiated event titles and narrative descriptions. While we considered constraining the agent to produce standardized schedule formats amenable to regex testing through few-shot learning, we ultimately rejected this approach as it would have compromised the agent's ability to provide contextual information about events when users made follow-up inquiries (such as "who is attending this meeting?"). We prioritized preserving the agent's conversational flexibility and comprehensive response capabilities over simplifying our testing methodology.

3.2 Testing Methodology

To overcome our testing challenges, we implemented an evaluation framework utilizing a separate "evaluator model" that assesses the accuracy of our scheduling agent's outputs against predefined ground truth test cases. This approach elegantly solved our testing dilemma by allowing the agent to maintain its conversational flexibility and diverse output formats while still enabling rigorous validation of its scheduling capabilities. Rather than forcing our agent to conform to easily testable but restrictive output schemas, the evaluator model applies semantic understanding to determine if the generated schedules contain all required information regardless of presentation format. By decoupling generation from evaluation, we created a more robust testing methodology that preserves the agent's ability to handle natural follow-up questions about event details. Below is an example from our [test.yaml](#) configuration file demonstrating this approach:

```
- user_messages:
  - What meetings do I have on June 2nd?
  expected_result: "There are no meetings on June 2nd / I don't know"
```

Our test script will run the user messages into the agent, take the output and feed it to the evaluator model, prompting it to compare the agent’s output to the expected result (see Figure 3).

```
Compare the output from an LLM scheduling assistant with the expected results of a user query. The expected result contains the relevant details or response that the LLM should provide.
The format of the assistant's output doesn't matter much, but the content should convey the same message. Times, dates, and locations should match exactly. The year is assumed to be 2001 if not stated.
It's OK if the agent outputs additional details, as long as the important details (time, date, location) match.
Return 'true' or 'false' along with a 1-2 sentence explanation.

Expected result: {expected_result}

Assistant Output: {stdout}
```

Figure 3: The prompt used for evaluating our agent’s responses.

3.3 Quantitative Evaluation

We constructed a total of 12 test cases ranging from simple queries to more complex multi-turn interactions, and even some that tried to trick the agent into hallucinating. We tested both Llama-3.1-8B-Instruct with our graph-based router approach and GPT-4o mini with our ReAct approach. For evaluation, we use the dataset of 154 emails from Kay Mann’s inbox described in section 2.1, as well as a subset of that dataset consisting of only meeting-related emails, of which there are 114. Both models were configured to generate with a sampling temperature of 0.4. We ran each test 6 times and obtained the results in Table 1.

Model / Architecture	Dataset	Pass@1
Llama 3.1 8B / Graph	Meeting-related only	62%
Llama 3.1 8B / Graph	Full inbox	15%
GPT-4o mini / ReAct	Meeting-related only	90%
GPT-4o mini / ReAct	Full inbox	70%

Table 1: Evaluation Results for our benchmark.

Based on our results and the model responses logged during testing, we observed that Llama struggles when trying to answer follow-up questions, leading to a much lower success rate. It often fails to use the retrieval tool even with the router implemented. GPT-4o mini, on the other hand, performed quite well, even when the inbox included emails not related to meetings and scheduling.

3.4 Qualitative Evaluation

Writing automated tests for our agent was difficult and likely did not capture the full picture of the agent’s capabilities. We thus spent a lot of time interacting with the agent ourselves and

verifying its outputs. Figures 4-6 show some examples of the conversations we've attempted.

```
User (blank to quit): give me my flight info
===== Ai Message =====

Your flight information is as follows:

- Outbound: June 4, 2001, from Houston, TX to Memphis, TN, departing at 7:45 AM and arriving at 9:30 AM (Flight CO 4096, Seat 11C).
- Return: June 4, 2001, from Memphis, TN to Houston, TX, departing at 5:55 PM and arriving at 7:40 PM (Flight CO 3841, Seat 12C).

This is a ticketless reservation, so please have a picture ID available at the airport.
User (blank to quit):
```

Figure 4: A user requests their flight information, without specifying a date. The model is able to find the only flight information mentioned in the email inbox and returns all relevant details.

```
User (blank to quit): when is paiges birthday
===== Ai Message =====

**Paige's Birthday:**
- **Date:** November 2, 2023 (Thursday)
- **Event:** 21st Birthday Celebration
User (blank to quit): when did I take off
===== Ai Message =====

**Days Off:**
- **Date:** November 2, 2023 (Thursday)
  - **Event:** Day off for Paige's 21st birthday celebration
- **Date:** November 3, 2023 (Friday)
  - **Event:** Day off
```

Figure 5: A user asks about information on someone's birthday, and follows up with a question related to the birthday. The model is able to find and get the date of the birthday and connect that the day off is related to it as well.

```
User (blank to quit): could you tell me what my day looks like one June 1st?
===== Ai Message =====

**Schedule for June 1, 2001:**

- **12:45 PM:** Lunch meeting for a new lawyer at a restaurant (location not specified).

If you need any more information or have other questions, feel free to ask!
User (blank to quit): anything else to do on june 1?
===== Ai Message =====

**Additional Notes for June 1, 2001:**

- There is a reservation for lunch at 12:45 PM for a new lawyer in your group.
- You may want to confirm the location of the lunch, as there was a discussion about finding a non-smoking option.

No other specific tasks or events were found for this date in the retrieved context. Let me know if you need anything else!
```

Figure 6: A user asks about a schedule on a specific day and follows up inquiring about any additional information. The model is able to first get the scheduled item of the day and answer the follow up question about anything that there is needed to know about that day.

Our qualitative assessment revealed that while our implementation demonstrated robustness when faced with unusual tasks, the models exhibited significantly different behavior in real-world use. Llama 3.1 8B frequently struggled with utilizing the retrieval tool despite our graph-based router implementation, often failing to retrieve relevant information when needed. This resulted in disjointed interactions where users would need to repeat themselves or rephrase

questions to get appropriate responses. In contrast, GPT-4o mini excelled at maintaining conversational continuity, consistently accessing the retrieval tool when required.

4. Discussion

4.1 Early Attempts

Our initial goal was to use our own emails as the dataset instead of relying on the Enron corpus. We explored the possibility of downloading personal email data into a CSV format, focusing specifically on our Northeastern Outlook accounts. However, this approach quickly proved impractical. Exporting emails from Outlook would have required administrative permissions which were outside the scope and timeline of our project. As a result, we pivoted to using the publicly available Enron email dataset.

4.2 Alternative Approaches and Future Work

Due to time constraints, we were unable to implement several ideas that we had initially considered. This section outlines a few of those directions, which could be explored in future iterations of the project.

4.2.1 Google Suite Integration

We originally envisioned integrating our agent with Gmail and Google Calendar, allowing the system to directly access a user's inbox and calendar. While this feature wasn't fully realized, we did prototype the ability to automatically add scheduled events to Google Calendar. However, we didn't have time to fully test it, so we omitted this from our final presentation. In the future, we'd like to build on this by linking directly to a live inbox, enabling the agent to work with real-time email data and offer more personalized scheduling support.

4.2.2 Improving Llama Integration

Our integration with the Llama model was somewhat unconventional. Unlike GPT-4, which was able to follow a standard ReAct-style agent loop, Llama required more manual prompting and routing. If we were to continue this project, we would explore fine-tuning the Llama model to better support tool usage within a conventional agent framework. Despite these limitations, Llama occasionally produced strong responses when it functioned correctly. With better tool invocation and more consistent behavior, its performance could likely be improved significantly.

5. Code, Benchmark, and Dataset

Our full code can be found at: <https://github.com/BrandonLim8890/email-scheduler>

The initial dataset of Enron emails was downloaded from:

<https://www.kaggle.com/code/gpreda/parse-and-process-enron-emails-dataset/input>

Our automated benchmark can be found at:

<https://github.com/BrandonLim8890/email-scheduler/blob/main/tests.yaml>