慶應義塾大学学術情報リポジトリ
Keio Associated Repository of Academic resouces

Title	Assist scheduling based on multi-platform emails using a large language model
Sub Title	
Author	チョウ, 水鈺(Zhao, Shuiyu) 加藤, 朗(Kato, Akira)
Publisher	慶應義塾大学大学院メディアデザイン研究科
Publication year	2025
Jtitle	
JaLC DOI	
Abstract	
Notes	修士学位論文. 2025年度メディアデザイン学 第1203号
Genre	Thesis or Dissertation
URL	https://koara.lib.keio.ac.jp/xoonips/modules/xoonips/detail.php?koara_id=KO40001001-00002025-1203

慶應義塾大学学術情報リポジトリ(KOARA)に掲載されているコンテンツの著作権は、それぞれの著作者、学会または出版社/発行者に帰属し、その権利は著作権法によって保護されています。引用にあたっては、著作権法を遵守してご利用ください。

The copyrights of content available on the KeiO Associated Repository of Academic resources (KOARA) belong to the respective authors, academic societies, or publishers/issuers, and these rights are protected by the Japanese Copyright Act. When quoting the content, please follow the Japanese copyright act.

Master's Thesis
Academic Year 2025

Assist Scheduling Based on Multi-platform Emails Using a Large Language Model

Keio University
Graduate School of Media Design

Shuiyu Zhao

A Master's Thesis
submitted to Keio University Graduate School of Media Design
in partial fulfillment of the requirements for the degree of
Master of Media Design

Shuiyu Zhao

Master's Thesis Advisory Committee:

Professor Akira Kato	(Main Research Supervisor)
Professor Kazunori Sugiura	(Sub Research Supervisor)

Master's Thesis Review Committee:

Professor Akira Kato	(Chair)
Professor Kazunori Sugiura	(Co-Reviewer)
Associate Professor Shota Nagayama	(Co-Reviewer)

Abstract of Master's Thesis of Academic Year 2025

Assist Scheduling Based on Multi-platform Emails Using a Large Language Model

Category: Science / Engineering

Summary

Email remains a common medium for informal schedule coordination, yet traditional calendar tools often fail to capture natural language expressions of time. To address this, we propose a system that retrieves emails via IMAP, groups them by session, and extracts scheduling information using a hybrid method of regular expressions and large language models (LLMs).

The system features a calendar-based GUI and allows users to confirm or edit extracted events, with all results logged for privacy and feedback.

We also record the extraction results and allow reliability feedback. In the experiment, 12 participants used the system during a one-week period. The system extracted 17.2% more events on average than pattern-based methods during the experiment (74.9 vs. 63.9 events per user). The post-interaction analysis revealed that system-assisted extraction matched manual extraction accuracy for related events with a minimal difference of 3.0% in improvement rates.

These results suggest that combining LLMs with user interaction offers a practical solution for email-based scheduling.

Keywords:

email-based scheduling, content extraction, Large Language Models, user interaction design

Keio University Graduate School of Media Design

Shuiyu Zhao

Contents

List of Figures

List of Tables

Acknowledgements

I am indebt to Professor Akira Kato for guiding not only about research but with many aspects of my life.

Throughout the development of the system, he provided crucial support, especially in the implementation of the IMAP-based modules. His expertise proved essential for solving protocol-level problems and he even directly assisted me in finding suitable RFC references for technical thesis sections.

During the writing process, Professor Kato offered detailed chapter-by-chapter feedback, helping me refine both the structure and content of the thesis. Beyond academics, he was always warm and supportive. I still remember one of the most thoughtful moments: he even helped me plan the transportation for my trip to Shizuoka to attend a concert. Besides, before each meeting, he would ask me how about this week and chat with me about daily life. I also felt comfortable asking him questions beyond the scope of research. His support extended far beyond academics and meant a great deal during my time in graduate school.

I would also like to sincerely thank all the participants who took part in the user experiments. Their time and feedback were invaluable in shaping and validating the system, and their support made this research possible.

Chapter 1
Introduction

1.1. Background

Email remains one of the most widely used communication tools in professional, academic and personal Settings. More than 4 billion users worldwide rely on email for daily communication, a large part of which involves coordinating meetings, setting deadlines or confirming appointments.

Despite calendar applications such as Google Calendar or Outlook, many users still manage their schedules through informal email exchanges. This is especially true for scheduling between organizations or among users using different calendar platforms.

However, emails are essentially unstructured. Unlike calendar interfaces, they allow for free-form natural language input. Schedule information is usually embedded in ambiguous expressions such as “Next Friday” or “tomorrow”. This information may be scattered across multiple message and reply chains.

Traditional rule-based methods including regular expressions fail to handle the natural language variability and ambiguity found in emails. The systems fail to detect hidden meanings while also struggling to interpret unclear references when they lack understanding of context.

Recently, Large Language Models (LLMs) such as GPT-4 and Gemini have demonstrated strong performance in semantic parsing, intent recognition, and multi-round conversation understanding. These models have achieved impressive results in various tasks by leveraging context learning without fine-tuning in specific domains.

The emerging trends indicate that systems need to unite LLM interpretive abilities with dependable email data access to deliver lightweight scheduling support which aligns with user workflows.

1.2. Motivation

As digital communication intensifies, users increasingly experience cognitive overload, especially in managing messages across multiple platforms. Kushlev and Dunn [1] demonstrated that frequent email checking leads to heightened stress levels, emphasizing the psychological burden associated with high-volume information streams. Their findings highlight the need for systems that reduce the frequency and cognitive load of email interactions.

Despite the emergence of various collaboration tools, email remains a primary medium for both formal and informal coordination. Grevet et al. [2] analyzed the evolving role of email as a tool for social task management, noting that many users rely on email threads to track pending actions, delegate responsibilities, and informally negotiate schedules. However, conventional email clients lack built-in mechanisms to support such task-oriented workflows, requiring users to manually manage dates and intentions buried in free-form text.

To address these limitations, earlier studies have proposed intelligent systems that assist users in managing schedules through natural language. Berry et al. [3] introduced PTIME, a personalized scheduling assistant that learns user preferences and supports negotiation by extracting constraints and time proposals from emails. This research illustrates the feasibility of using AI techniques to semi-automate scheduling while preserving user control and flexibility.

Together, these studies provide a foundation for the development of systems like ours, which aim to support users in processing natural language schedule coordination through email—reducing stress, improving accuracy, and enhancing task visibility.

1.3. Challenges in Email-based Schedule Extraction

Although email remains the primary tool for coordinating meetings and managing schedules, extracting relevant information from free-format email threads poses some challenges.

First of all, due to the scattered and implicit nature of the schedule discussion,

users often miss important events. Key information may be hidden in long threads, vaguely worded, or embedded in forwarded content.

Secondly, when multiple proposals are discussed without a clear consensus record, especially in multi-party dialogues, schedule conflicts may arise. This makes it difficult for both users and the automated system to determine the final agreed date or time.

Thirdly, existing tools often lack the accuracy to reliably extract dates, times and intentions from natural language. Systems based on regular expressions may miss the context.

Finally, current systems rarely offer personalized processing based on a single user. Users may prefer different formats, but most assistants offer limited control over how to extract and display schedule data.

These issues highlight the need for more user-aware methods for email-based schedule extraction, which has inspired the research directions outlined in the following sections.

Chapter 2
Related Works

2.1. Existing Strategy for Email Retrieval

2.1.1 Keyword-Based Filtering

Keyword-based filtering has long served as a baseline for extracting task-relevant content from emails, particularly in domains such as spam detection, classification, and information retrieval.

Earlier methods typically rely on manually defined keyword lists or statistical measures (e.g., TF-IDF [4]) to identify important terms. TF-IDF quantifies how important a word is to a specific document relative to a larger corpus. It increases proportionally with the number of times a word appears in the document but is offset by the frequency of the word across all documents, thus highlighting rare but informative terms. In early email classification and filtering systems, TF-IDF was commonly applied to identify relevant documents or extract salient phrases. For example, emails containing high-weighted terms or specific dates are often prioritized for schedule-related analysis. TF-IDF also serves as an efficient feature selection method in unsupervised settings, requiring no labeled data. However, TF-IDF has limitations. It ignores the semantics and syntactic structure of text, which can lead to low recall in contexts where key information is expressed implicitly. Additionally, it struggles with synonymy and polysemy, making it less effective in informal or multilingual communication. While straightforward and computationally efficient, TF-IDF suffer from low recall and poor generalization in real-world communication, where relevant information is often conveyed implicitly or through diverse linguistic forms.

To address these limitations, more recent work has proposed some other enhanced keyword extraction techniques. For example, Azam et al.(2025) inves-

tigate hybrid approaches that use TextRank to identify salient terms, and then refines them using bi-directional GRU or BERT models trained on data [5]. These models effectively capture contextualized keyword relevance by integrating local dependencies from neural encoders with global term importance derived from graph centrality.

However, such methods still treat keyword extraction and filtering as shallow tasks - they focus on word-level importance rather than deeper semantic meaning or communicative intent. In the context of schedule-related emails, critical phrases such as "we should catch up after the 15th" often lack overt keywords like "meeting" or "appointment", making them difficult to detect using keyword-based rules alone.

Our research builds on this gap, using intent and context-aware reasoning from Large Language Models (LLMs) based on word-level keyword filtering. By using semantically rich templates and demonstration prompt LLMs, we are able to identify implicit scheduling cues regardless of lexical form, achieving improvement from keyword detection to semantic understanding.

2.1.2 Structured Scheduling Data and Its Constraints

With the wide adoption of electronic calendar systems, the standardized format for representing and exchanging schedule information has become crucial. Native support for major applications such as Gmail and Outlook. The ics file format is based on the iCalendar standard (RFC 5545) [6]. This standard defines a structured plain text format for describing events, to-do items, idle/busy information, and log entries in a platform-independent manner. By encoding key attributes such as VEVENT, DTSTART, DTEND and SUMMARY using well-defined ABNF syntax, it enables interoperability across email clients and scheduling services.

These applications typically use `.ICS` attachments to exchange scheduling information via email. For example, when sending a meeting invitation via Gmail or Outlook, the system usually attaches an `.ICS` file, which can be parsed and presented by the calendar interface of the recipient (Figure 2.1). This format is MIME-compatible, supports UTF-8 character encoding, and allows integration with protocols such as SMTP or HTTP for transmission. Strong support for cir-

cular events (RRULE), time zone specifications (TZID), and alert notifications (VALARM) makes.ICS the preferred medium for consistent scheduling representations across heterogeneous systems.

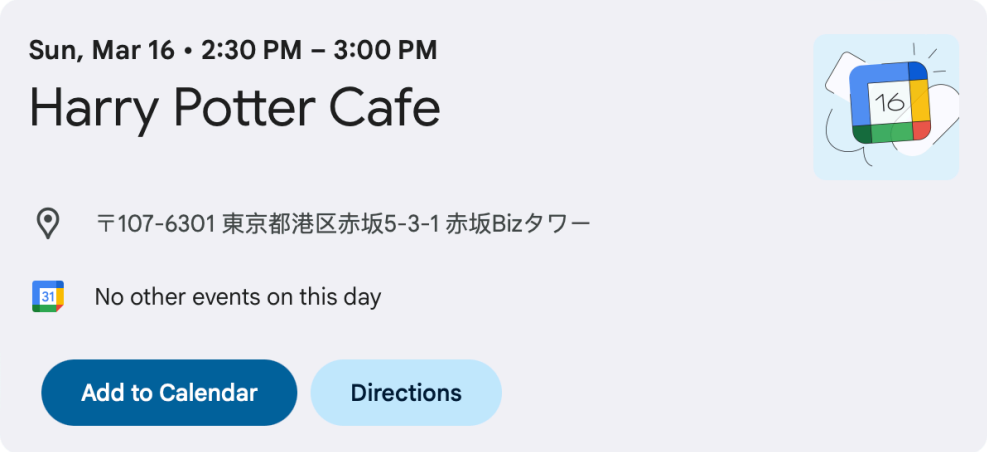

Figure 2.1 An Example of a Calendar Invitation Email Including ICS Format

In addition to traditional attachment-based methods, modern email platforms are increasingly adopting interactive content formats such as AMP for Email [7]. Developed by Google, this framework extends standard HTML with dynamic components that allow users to interact with email content directly, such as responding to event invitations, filling out forms, or viewing up-to-date calendar information, without leaving their inbox. For example, while an AMP email might contain an event invitation, a user clicking an "Yes" button in an AMP email completes the RSVP action shown in Figure 2.2. AMP emails use the MIME type `text/x-amp-html` and must be bundled with fallback HTML and plain-text versions to ensure compatibility across email clients. By embedding structured data and leveraging real-time content updates, AMP for Email offers a more responsive alternative to static ICS attachments. While not yet universally supported, this approach represents a shift toward richer, integrated scheduling experiences within the email ecosystem.

However, in real-world communication, a significant portion of scheduling information is not embedded in structured `.ICS` or `text/x-amp-html` files but is transmitted through free text content located in various parts of the email body,

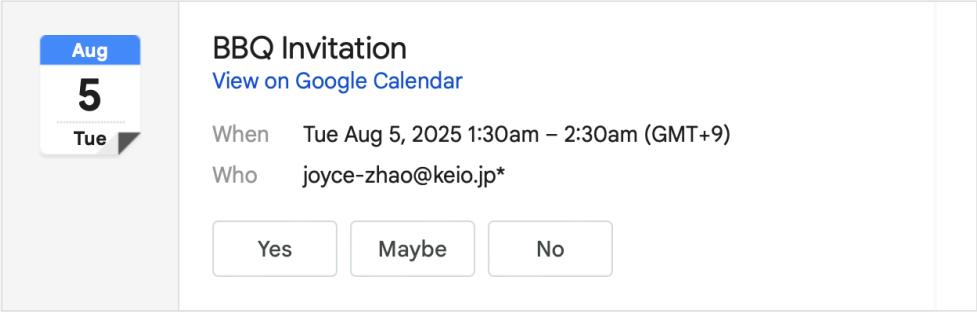

Figure 2.2 An Example of a Calendar Invitation Email Including Response Options

such as informal proposals, confirmations, or embedded forwarding content. Existing clients often fail to capture such implicit events.

To address this gap, our system utilizes Large Language Models (LLMs) and logic based on regular expressions to extract event-related information from all parts of the email body. Unlike traditional calendar applications that only rely on RFC-compatible `.ICS` data, our system can identify information related to plans regardless of its structure or position in the message. This hybrid approach enables more comprehensive and personalized scheduling support, especially in environments involving multilingual communication or negotiation-based coordination.

2.1.3 Smart Features in Google Workspace

Google Workspace provides built-in support for integrating calendar functionalities directly within Gmail, streamlining the process of sending and managing calendar invitations via email [8]. As administrators can configure Gmail settings to automatically convert compatible messages into actionable calendar invites, enabling seamless scheduling across users within the organization, as shown in Figure 2.3 [8].

This integration leverages email headers and standardized invitation formats to populate Google Calendar events, offering features such as RSVP tracking (Figure 2.2), guest management, and conflict detection. By embedding structured event metadata within the email transmission, the system supports interoperabil-

ity across devices and platforms without requiring users to manually extract or interpret scheduling details.

This mechanism highlights the increasing shift toward automation in email-driven calendar workflows, and underscores the need for intelligent systems capable of understanding and processing implicit and explicit scheduling signals in email threads.

Smart feature setting	Where to find the setting	Description	Experience example
Smart features in Gmail, Chat, and Meet	Gmail	Controls use of your Gmail, Chat, and Meet data to provide smart features for these products.	Smart Compose in Gmail Summary cards
Smart features in Google Workspace	Gmail Calendar* Chat* Drive* Meet*	Controls use of your Workspace data to provide smart features across Workspace products. This setting does not affect smart features that personalize your experience in Gmail, Chat, and Meet, which are subject to the control described above.	Show events from Gmail in Calendar, such as flight itineraries and invitations Google Workspace with Gemini
Smart features in other Google products	Gmail Calendar* Chat* Drive* Meet*	Controls use of your Workspace data to provide smart features in other Google products. Data sharing and personalization in certain Google products may be subject to additional controls, such as Web and App Activity and DMA linked services (in the EU).	Restaurant reservations and to-go orders in Maps Suggested tickets, passes and loyalty cards in Wallet

* You can only turn on this setting from a web browser.

Figure 2.3 Smart Features in Gmail Example for Custom Scheduling

While Google Workspace provides smart features that automatically surface schedule-related content, such as flight itineraries and meeting invitations shown in the Experience example column of Figure 2.3, within applications like Gmail and Calendar, these capabilities remain largely dependent on predefined formats and structured data. As shown in the system documentation, smart features rely on the integration of Workspace services and require centralized control through browser-based settings. However, they do not generalize well to loosely structured

emails, nor do they offer user-level customization in terms of event interpretation or prompt adjustment.

2.2. Use of Large Language Models (LLMs)

2.2.1 Text Mining

Text mining is a core subfield of Natural Language Processing (NLP) that focuses on extracting meaningful and structured information from large volumes of unstructured textual data. Common tasks in text mining include information retrieval, document classification, sentiment analysis, and named entity recognition (NER), all of which aim to bridge the gap between raw language data and actionable insights.

In recent years, the emergence of Large Language Models (LLMs) has revolutionized how text mining is performed. Unlike traditional models that rely on handcrafted features or shallow statistical methods, LLMs leverage massive pretraining on diverse corpora to capture rich contextual semantics, enabling a broader understanding of human language. A comprehensive survey by Yang et al. [9] provides an in-depth review of recent progress in NER, highlighting the shift from traditional feature engineering approaches to neural architectures such as BiLSTM-CRF, and subsequently to transformer-based models including BERT, RoBERTa, and GPT-based variants.

The survey categorizes existing methods into three main paradigms: sequence labeling, span-based classification, and prompt-based frameworks. Among these, prompt-based NER methods have emerged as a promising solution for adapting LLMs to low-resource or domain-specific tasks, leveraging in-context learning and instruction tuning to generalize without extensive fine-tuning. Furthermore, the paper discusses how LLMs can perform zero-shot or few-shot extraction when guided by carefully designed prompts, enabling more flexible and scalable entity recognition in diverse text sources.

These insights directly inform our work, where we apply LLMs to extract temporal and actionable information from email conversations. Unlike conventional NER tasks that focus on predefined entity types (e.g., PERSON, LOCATION),

our use case requires identifying contextual scheduling expressions such as "sometime after next Friday" or "reschedule our presentation." Inspired by the advances surveyed in [9], we adopt a hybrid approach combining prompt engineering with email thread context modeling, aiming to extend NER techniques into the domain of real-world event extraction from informal, user-generated messages.

2.2.2 Semantic Parsing

The extraction of structured data from unstructured natural language inputs relies heavily on semantic parsing. The traditional parsing methods require rule-based grammars and statistical parsers which need extensive supervision and domain-specific design. Deep learning models especially Recursive Neural Networks (RNNs) have improved the ability of systems to learn both syntactic and compositional semantic representations within a single framework.

Socher et al. (2011) introduced recursive neural networks to perform joint syntactic and semantic parsing of natural language sentences. The model learns to convert words and phrases into continuous vector spaces before constructing hierarchical structures that show both grammatical correctness and semantic interpretation. The method achieves top-tier performance in syntactic parsing without requiring predefined grammar rules. The learned representations successfully extract hidden meanings across different phrase levels which proves that neural models can perform end-to-end semantic understanding [10].

The research supports a main goal of this study because neural models demonstrate the ability to extract deep semantic structures from unprocessed text therefore schedule-related semantics should also be extractable. This research expands upon the original concept by implementing modern LLMs for both general semantic interpretation and specific extraction of semantic schedules from free-form email content which traditional methods (regex or keyword filters) cannot effectively process because of context sensitivity and linguistic variability.

2.2.3 Contextual Understanding and Intent Recognition

The interpretation of scheduling-related communication depends heavily on understanding user intent especially when the intent remains hidden or uses informal

language. GPT-3 demonstrates the ability to perform various tasks through in-context few-shot prompting without needing additional fine-tuning [11].

Wang et al. (2023) showed that LLMs successfully detect user intent in mobile graphical user interfaces (GUIs) through their ability to create questions and summarize screens and provide answers while converting instructions into interface actions. The model demonstrates its ability to understand user objectives through its performance across different structured interaction tasks.

Inspired by this, our work applies LLMs to the domain of email-based schedule management, where intent recognition involves understanding user instructions, confirmations, or vague commitments (e.g., "let's meet sometime next week" or "how about Tuesday afternoon?") from unstructured email threads. These tasks in the UI environment require extensive contextual understanding because they need to interpret both individual sentences and the complete conversation history, sender information, and email reply functions. Our system improves the performance of LLMs in extracting details of actionable events by implementing time-sensitive LLMs inference. The system provides clear negotiation support by detecting intent changes in the email thread, including meeting proposals, rescheduling and confirmation processes.

2.3. Standardized Access and Structuring of Email Data

2.3.1 Unified Access Across Email Platforms

Managing schedules across multiple email platforms, such as Gmail, Outlook, and university, hosted systems—requires unified access to messages stored on various servers. The Internet Message Access Protocol (IMAP), especially its modern revision IMAP4rev2 as specified in RFC 9051 [12], provides a standardized mechanism for clients to access and manipulate email messages on remote servers. IMAP4rev2 supports essential operations such as searching, selective fetching of message parts, mailbox manipulation, and synchronization of offline changes, enabling consistent email handling across different providers and devices.

IMAP's design abstracts away the specifics of each email service, allowing appli-

cations to fetch and organize messages using consistent commands and identifiers such as UIDs and message sequence numbers. This makes it well-suited for building applications that aggregate content from heterogeneous sources into a unified view. Furthermore, IMAP enables fine-grained access to message metadata (e.g., flags, subjects, timestamps) and supports concurrent mailbox access, which is critical for responsive multi-threaded email applications.

In our system, IMAP serves as the foundation for building a unified mailbox that spans multiple email platforms. By supporting login and message retrieval across Gmail and university servers, our tool enables users to centralize their scheduling-related emails without migrating accounts or relying on vendor-specific APIs. This IMAP-based integration ensures compatibility, scalability, and minimal user-side setup, while also allowing downstream processing, such as schedule extraction or thread grouping, to operate on a unified dataset. Compared to systems that rely on local mailbox exports or single-platform APIs, our approach offers greater flexibility and real-time access to live email data, supporting timely scheduling decisions.

2.3.2 Thread Tracking

Email-based scheduling usually involves multiple rounds of negotiations, where participants propose, modify and confirm event times across multiple messages. Accurately tracking these asynchronous conversations is crucial for understanding the evolution of scheduling decisions. A key mechanism to achieve this purpose is to use the standardized message header in RFC 2822 [13], which defines the structure of ARPA Internet text messages, including unique identifiers such as `Message-ID`. As well as the `In-Reply-To` and `References` headers. These fields can reconstruct the email thread by linking the replies to their parent messages in a directed acyclic graph structure.

Modern email clients like Gmail and Outlook use these headers to visually group emails into session threads. However, this kind of visual thread does not necessarily translate into a logical thread-level understanding of the scheduling context. Especially in multilingual, informal or forwarded communication, negotiation intentions and confirmations can be scattered among several loosely connected messages.

Our system addresses these challenges by using `Message-ID` and the headers related to responses as the baselines for grouping emails, while applying additional semantic analysis using LLMs to determine whether the messages belong to the same scheduling thread. This hybrid thread tracking method can handle complex negotiations robustly, including situations where participants reply across multiple chains or contain forwarded content. By combining the structural metrics of RFC 2822 and content-based cues, our system ensures a more reliable aggregation of multi-round scheduling discussions. This is particularly valuable in scenarios where it is confirmed to be implicit or scattered, which is very common in the coordination of the real world.

Chapter 3
Proposal

3.1. Current Issues in Email-Based Scheduling

In the digital age, mobile communication technologies have led to a significant increase in the amount of information people receive every day. Among these, email remains still a primary channel for professional daily coordination. However, the quantity and uncertainty of email content, from important meeting invitations to promotional information, pose major challenges to effective scheduling.

Although spam filtering technologies can remove some obviously irrelevant content, there is still a considerable proportion of non-SPAM emails that are not entirely useful. Even within the category of non-SPAM, users need to distinguish between messages that require attention and those that do not. This has caused many problems for seemingly simple coordination between checking emails and scheduling based on email content.

This phenomenon is commonly referred to as "information overload" [14], and it significantly affects the ability of users to extract and process scheduling-related information embedded in the email thread, resulting in manual scheduling being an error-prone and time-consuming process.

3.1.1 Missed Events

One of the most common problems with email-based scheduling is the omission or mistake of important events. Due to information overload, users will receive a lot of non-important information unrelated to their schedules, which makes it very easy for them to overlook important schedule arrangements. Even when a user has opened and read a message related to an event, due to the lack of immediate action, such as transferring it to the calendar, the message may still be forgotten.

Such problems may arise in daily life, study, or work. For instance, students may receive messages from professors, administrative offices, classmates, etc. Company employees may also receive a large number of communications from colleagues, superiors and collaborators, all of which may contain time-sensitive scheduling information.

3.1.2 Schedule Conflicts

Secondly, schedule conflicts often occur due to the lack of a conflict checking mechanism for different emails. Since most email systems handle each message separately, there is no way to detect whether two meeting times from different proposals conflict with each other. Therefore, users may agree to appointments with overlapping times. A wrong commitment may result in failure to attend as scheduled or even more serious consequences. Unlike centralized scheduling systems, such as shared calendars or team collaboration tools, email-based scheduling lacks mechanisms to resolve these conflicts. Users have to manually check their emails and calendars by themselves, which is not only inefficient and time-consuming, but also prone to errors.

3.1.3 Lack of Accuracy and Usability

Although existing email services attempt to automatically extract events from emails and offer "intelligent suggestions" to add them to the calendar that comes with the emails themselves, these functions are still limited in terms of accuracy and usability. For example, events without an absolutely definite date (e.g., "Let's meet at noon next Monday") are usually ignored. For another example, in Gmail, files lacking standard calendar attachment formats (e.g., `.ics` files from websites registered for Gmail [6]) are usually not extracted correctly, even if the email already contains clear date and time information. These shortcomings have reduced the usability of the existing email service automation functions for users.

Another significant limitation of the existing email services tools is that they are unable to effectively integrate emails from different service providers (e.g., Gmail, Outlook, and institutional-provided mail servers) into a unified calendar interface. Most current systems are designed to operate within a single platform, which

makes it difficult for users who rely on multiple accounts for academic, professional and personal communication to manage their schedules comprehensively. For instance, an individual can use the email provided by the university for course study and meetings, the email provided by the company for internship or job applications, and the personal Gmail account for socializing or event notifications. If there is no centralized integration, users have to manually cross-check and merge events from different mail services, thereby increasing the risk of ignoring conflicts or missing important updates.

3.1.4 Limited User Personalization

Another limitation of the existing email services is the lack of personalization and interactivity. Most existing email services adopt a unified functional approach, providing the same event extraction logic for all users regardless of individual differences. In fact, different users have different priorities to consider. For instance, students might give priority to scheduling emails from professors, while company employees might prioritize responses from superiors and fellows. If users cannot intelligently define and interact with the event extraction logic, the events extracted by the email service cannot accurately reflect the true priorities of users.

3.2. Research Questions

3.2.1 Accurate Extraction of Schedule Information

The subject of an email usually contains unstructured and implicit times and events, such as “Let’s meet next Thursday” or “There will be a meeting tomorrow”. Traditional systems based on rule-based structured patterns often fail to capture such natural language expressions. In addition, multilingual environments, including English, Japanese and Chinese emails, further affect the accuracy of accurate extraction due to the different date representations based on different languages.

This study aims to explore the effectiveness of combining rule-based pattern matching with Large Language Models (LLMs [15]) to improve the accuracy of scheduling related information extraction. Rule-based methods provide fast and interpretable results for structured patterns (e.g., `YYYY-MM-DD`), and LLMs can

better understand the context and infer implicit scheduling cues. Through the two methods of individual evaluation and combined evaluation, the goal is to develop a hybrid framework to achieve high precision among different language styles and message types.

3.2.2 Negotiation-Aware Scheduling

Email-based scheduling sometimes involves multiple rounds of back-and-forth communication, especially when coordinating events among several participants. These negotiations are usually distributed in threads composed of multiple emails. The proposed candidate date, the preferred time for updates after negotiations, and the time of the final confirmation event typically appear in different email messages. Traditional email service tools lack the ability to handle such conversation threads and process each message in isolation, resulting in conflicts in the time manually arranged by users.

This study explores how intelligent assistants can identify, group and analyze scheduling negotiations by using message threads (e.g., through `Message-ID` [13]) and context understanding through Large Language Models (LLMs). The goal is to help users track ongoing negotiations, intelligently obtain the final determined time result through context understanding, and avoid inconsistent or overlooked responses. By becoming negotiation-aware, this intelligent email assistant can support more reliable and user-friendly coordination in complex dialogue thread scheduling scenarios.

3.2.3 Cross-Platform Email Integration

Many users operate multiple email accounts on different platforms, such as email addresses provided by universities, personal Gmail accounts, and perhaps work emails. However, most of the existing scheduling tools are based on a single specific platform and are unable to aggregate emails from different providers into a unified calendar. This dispersion forces users to manually synchronize their schedules on multiple platforms, which is both time-consuming and error-prone.

This study addresses the technical and privacy challenges of building a cross-platform email scheduling assistant, which uses IMAP (Internet Message access

Protocol [12]) to obtain emails from various providers without changing their read/unread status. The proposed system operates entirely on a per-user basis, storing data locally and avoiding the privacy and security issues of centralized integration. By achieving unified access and analysis while protecting privacy and usability, this study aims to reduce the event time scheduling load of users with multiple inboxes.

3.2.4 Personalized and Interactive Schedule Filtering

The current email-based automated scheduling solutions often lack personalized user interaction. Users cannot control the logic by which the system automatically filters or prioritizes events in emails, nor can they specify what types of events the system is most important to them. Therefore, the suggestions for automation may be irrelevant or inaccurate, further leading to information overload.

This study explores how to personalize the event extraction process through the interactive prompt customization and user-oriented filtering mechanism of LLMs. For example, users can provide prompt instructions to emphasize events from specific senders, such as professors, or exclude social invitations that they do not care about or have nothing to do with the users. Through this interaction, the assistant becomes more intelligent, better meets the needs of individual users, increases trust, and reduces the manual workload required to manage email-based schedules.

3.3. Proposed Approach

3.3.1 Hybrid Extraction using Rules and LLMs

To improve the accuracy of extracting information related to the plan, this system combines rule-based pattern matching with Large Language Models (LLMs). Rule-based systems capture dates with clear formats (e.g., `2025-06-01`, `June 15th`), while LLMs handle more ambiguous or natural language expressions. By applying the two methods in parallel and comparing the results, the system can extract more comprehensive and reliable scheduling data from different email formats.

3.3.2 IMAP-Based Multi-Account Integration

In order to achieve the unified scheduling of different email services, this system enables multiple accounts, such as Gmail and universities email, to access the inbox and sending folder simultaneously through IMAP. Obtain emails without changing their read/unread status and store them locally for analysis. This enables a consistent scheduling assistant to respect user privacy and allow events to be detected from all relevant email sources in one place.

3.3.3 Thread-Level Schedule Negotiation Handling

This system uses the string values `Message-ID` and `In-Reply-To` to group the relevant emails into threads, allowing it to regard a series of negotiation messages through emails as the context of a single schedule. This enables it to identify coordination regarding the same event from the inbox and the outbox, such as dates, confirmations, and cancellations proposed by multiple participants. This assistant only displays the final negotiation results of the decision to help users manage conversational email-based schedules more accurately.

3.3.4 Prompt-Based User Interaction and Filtering

Users can interact with the assistant's analysis by entering prompts for customization (e.g., "No payment information is needed" or "Ignore emails from xxx"). These prompts are dynamically merged into the LLM prompt for more relevant and personalized extraction. The prompt history can be reused or adjusted to enable the system to understand user preferences and reduce irrelevant suggestions.

3.3.5 Privacy-Preserving Local Processing and Calendar Updates

To ensure user privacy and data security, all email analysis and event extraction are performed locally on the user's device. Do not save email content, metadata or scheduling data to any external central server. The extracted events are saved in the local JSON log file, which can serve as both a log file and the basis for calendar updates. Users can manually check and modify this log file as needed, for

example, by deleting sensitive personal information or editing private information before exporting data. This mechanism allows users to safeguard their privacy according to their own preferences.

3.4. Limitation

3.4.1 Non-SPAM, Email-Only Event Sources

This system restricts its event extraction process to email content. Communications from other platforms (such as chat applications (e.g. LINE, Slack), shared calendar tools, or other messaging services) have been excluded. This design focuses more on studying the related issues of planned events extracted from email services.

Furthermore, only non-spam emails are regarded as effective sources for event extraction. Spam is filtered out either by relying on the built-in spam classification of the user's email provider or through content-based exclusion criteria.

Although this filtering limits the relevance and accuracy of event extraction, it brings the risk of missing valid scheduling information. Information about time scheduling may be incorrectly classified as SPAM by the email service itself. Nonetheless, this design focuses on using non-spam as the sole data source, which is consistent with the goal of creating a lightweight and privacy-friendly assistant during the initial development stage.

3.4.2 IMAP-Only Protocol Support

The current system only supports accessing emails through the IMAP standard. IMAP supports the selective retrieval of email headings and bodies while retaining the message status (e.g., read/unread), making it highly suitable for incremental acquisition and thread grouping based on email message data. This protocol is widely supported by major providers such as Gmail and institutional mail systems, and ensures compatibility with standard email clients.

Currently, other email access methods, such as POP3 (Post Office Protocol 3 [16]), are not supported POP3 operates on a different example — it usually downloads all the messages to the local machine and then deletes them from the

server. Although this method may not retain the thread structure or message state, it is very valuable in scenarios where server access is restricted or archived data needs to be processed offline.

However, the future expansion of this system may explore the ability to analyze locally stored email files, such as those exported from email clients `.mbox` or `.eml` file. This will allow the assistant to operate in a completely offline environment. In the future revision of the prototype, the extension protocol will be considered to enhance compatibility and support for a broader user environment.

3.4.3 Assistant-Only Role

This system is clearly designed as an email-based schedule assistant rather than an autonomous scheduler. It does not make any decisions on behalf of the user, such as automatically accepting invitations, modifying calendar entries or automatically replying to emails related to scheduling. Its function is to extract potential event information and present it to users for confirmation and manual interaction operations. This design that only supports assistants ensures the retention of the user's main management and avoids the unexpected consequences brought by automatic behaviors, such as incorrect scheduling coordination, incorrect email responses, etc.

However, it also makes users take on more responsibilities to view and coordinate the extracted information and update their calendars accordingly. Although this restriction reduces the degree of automation, it enhances reliability and credibility, especially in professional field emails, as incorrect scheduling in such emails can have serious consequences.

In future versions, this assistant-only architecture may be extended with optional semi-automated actions (e.g., suggesting calendar drafts) while still preserving the user's final decision authority. This would allow the system to evolve toward greater efficiency while maintaining its user-centered design philosophy.

3.4.4 Per-User Isolation and Privacy

To protect user privacy and reduce security risks, the system strictly ensures that users can operate on sensitive information. The data of each user, the subject

line of the email and the rows related to the schedule, extracted events, and logs, are processed and stored locally and will not be shared in real time or sent to a centralized server. There is no cross-user synchronization or collaborative editing. This isolation can prevent data leakage and make the system suitable for an environment where emails contain any sensitive information.

However, this design also limits certain use cases, such as sharing team calendars, etc., which usually requires cross-user data visibility, allowing everyone in the team to synchronize the calendar. Future versions may explore optional privacy-protecting sharing mechanisms, but at the current stage, for simplicity and security, the isolation of individual users must be given priority.

3.4.5 Focus on Core Scheduling Without Fancy UI

In this prototype, the focus is on relying on reliable schedule extraction and logical data processing via email rather than implementing advanced calendar features. This system does not provide a fully interactive calendar interface, including drag-and-drop scheduling, visual conflict detection, or integration with external services such as Google Calendar or Outlook. On the contrary, it provides the lightest and most simplified visual representation of extracting events in the form of a calendar view, only used to help users view the identified time-related content. This decision reduces the complexity of development and ensures consistency across operating systems.

Although some users may expect modern calendar functions, excluding them can keep the system lightweight, privacy-protecting and more focused on the research of event extraction contained in email content. Once the underlying extraction logic is fully verified, advanced UI features can be considered in subsequent iterations.

Chapter 4
System Design

4.1. System Architecture Overview

4.1.1 Overall Pipeline Design

The system uses automated email extraction to obtain scheduling details from user accounts which it then displays through the calendar interface. The IMAP protocol retrieves all emails from both the `INBOX` and `SENTBOX` folders to provide a complete view of the conversation. The system groups received emails into threads through analysis of `Message-ID` and `In-Reply-To` headers to reconstruct and analyze the conversation context. The system applies regular expressions for structured patterns together with Large Language Models (LLMs) for context-aware interpretation to extract scheduling content within each thread. The system structures extracted information before applying user preference filters to produce a final output that integrates into the interactive calendar interface for event visualization and management.

4.1.2 Flow Diagram

The overall workflow is visualized as a flow diagram representing how raw emails are collected, analyzed, and presented to the user. The flow proceeds from email servers to local parsing modules, where both regex and LLM processes analyze the content. Then, structured events are saved to a log file and displayed in the calendar UI. Optional manual and automated update flows are also included in the diagram (Figure 4.1).

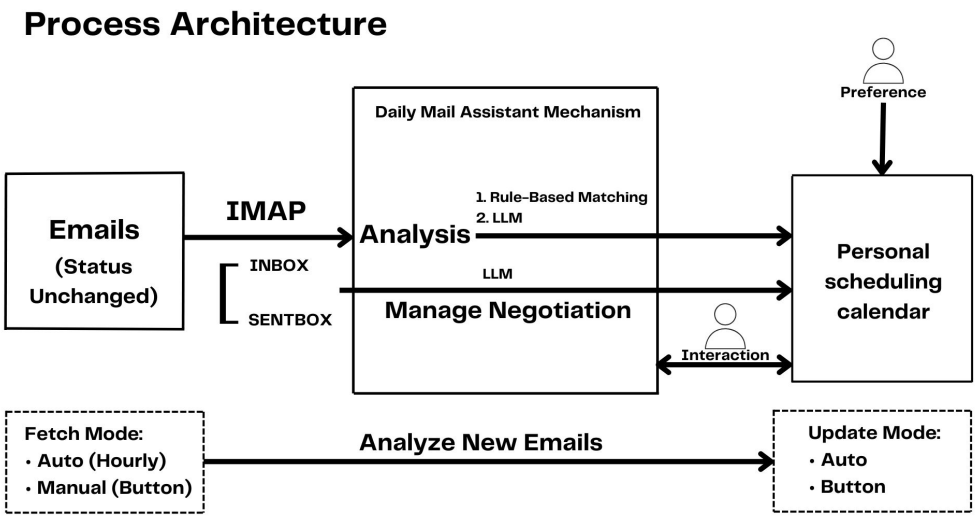

Figure 4.1 System Pipeline: From IMAP-based email fetching to event calendar visualization

4.2. Event Extraction Logic

4.2.1 Regex-Based Time Recognition

This module applies a set of handcrafted regular expressions to identify date and time expressions from email text. It captures formats such as “Jun 3rd, 12:00”, “2025/06/03”, and “03 Apr 2025”. The advantage of using regex is its speed and precision in extracting explicit time strings. These expressions are applied to both the subject and body of each email in the thread, ensuring comprehensive coverage. However, it may miss ambiguous or context-based dates or variations outside predefined templates.

4.2.2 LLM-Based Processing

To complement the limitations of regular expressions, we integrated the Gemini LLM API [17], which can interpret the context and infer scheduling information even if the time is indirectly referenced (e.g., “Let’s meet this Friday afternoon”). These phrases themselves are ambiguous and beyond the comprehension scope of purely rule-based methods.

The data sent to the LLMs includes the complete email thread and the set prompt, through which it is parsed into a structured format containing potential event candidates. The LLMs can understand the hidden time information through context analysis or coordinate the schedule of the information replied by multiple participants. It plays a crucial role in extracting events with concealment and ambiguity, which cannot be solved by regex alone.

To preliminarily verify the feasibility of our approach, we conducted an initial experiment by 500 synthetic emails. The results revealed that the prompt plays a critical role in guiding the LLM's performance. This finding suggests that iterative refinement of prompt construction is essential not only for improving the accuracy and consistency of event extraction, but also as a key component of overall system development. Therefore, it is crucial to incorporate prompt optimization as an integral part of the implementation step.

4.3. Email-to-Calendar Flow

4.3.1 Email Fetching and Thread Grouping

Use the IMAP protocol to obtain emails from the `INBOX` and `SENTBOX` folders to ensure complete bidirectional overwriting of conversations related to scheduling. This dual-source method captures both user-initiated and externally initiated messages, retaining the complete context of scheduling negotiation. To group messages into consistent threads, the system uses metadata headers such as `Message-ID`, `In-Reply-To` and `References`. Each email thread is reconstructed in chronological order by forming the relationship of messages through these data headers. This thread-based representation enables the system to distinguish multiple parallel conversations and track ongoing scheduling exchanges with high precision.

4.3.2 Multilingual Support for Scheduling

The system is designed to support multiple languages. This multilingual ability is indispensable in cross-cultural or international environments, such as in international universities or other international institutions and fields, where schedule

emails are written in various languages. The regex engine adopts region-specific rules to detect different date formats, such as “2024 年 12 月 5 日（木）”, “下午七点”, and “June 9, 2025 5:00 PM”. For LLM-based extraction, lightweight language detection routines analyze the content of emails and dynamically adjust the language of the prompts to match the detected language. This ensures that the Gemini API receives appropriate input from the context and improves the accuracy of extracting event candidates. The ability to handle multilingual content without user intervention enhances the usability of the system and expands its applicability.

4.3.3 Thread-Based Context Event Candidate Identification

Once a thread is built, the system will perform a comprehensive context analysis on each email thread to determine potential candidate events. This includes scanning the entire thread as a whole instead of processing individual messages in isolation, which allows the system to interpret the constantly changing scheduling discussions. The regex module will mark any explicit date or time expression. Meanwhile, the LLMs component is used to extract more subtle or implicit scheduling intentions, such as responses confirming or suggesting adjustments (e.g., “This works for me” or “I think we can change it to 10:00 a.m. tomorrow”). Then, the system analyzes these negotiations into a final candidate list result. And it stores relevant metadata, including the relevant thread ID, message source, title, body and receiving time.

4.3.4 Preference-Aware Event Selection

User preferences play a key role in refining the extracted event candidates into actionable calendar entries. Users can express dynamic preferences through natural language prompts. These prompts are directly input into the GUI, and submitted to the LLMs model along with the email thread, as shown in Figure 4.4.

During the reanalysis, the system incorporates these prompts into the reasoning process of the LLMs. LLMs analyzes the context of the email content based on the newly added prompts and adjusts the way of extracting or filter-

ing events accordingly. For example, the prompt "`Exclude emails from the sender no-reply@slack.com`" (Figure 4.4) will cause the system to ignore any events extracted from such sources. This mechanism provides users with a flexible and real-time way to control inclusion and exclusion rules without modifying the code or Settings.

By integrating personalized prompts into the scheduling pipeline, the system can be more closely aligned with the intentions of individual users and enhance personalization tailored to each user. In future versions, this feature may be expanded to include prompt history and implicit feedback, allowing the system to learn and improve user preferences over time.

4.3.5 Calendar Entry Generation and Integration

The final determined events are converted into structured entries and presented in the interactive calendar interface. Each entry includes key information such as the date and time, the source of the message, and a brief summary of the event. The calendar interface is implemented using a GUI based on tkinter, which supports the day view and the week view. Visual cues such as color coding and ICONS indicate event types, confidence levels, and editable capabilities. The system also maintains a persistent JSON-based event log to ensure synchronization across sessions and allow for retroactive modifications or deletions. This dual integration of the visual layer and the data layer provides transparency and user control.

4.4. Graphical User Interface (GUI) Design

4.4.1 Email Fetching via User Login

The graphical user interface provides a unified login window, supporting Gmail and Keio Mail, enabling users from different domains to use the system without modifying the code. First, prompt the user to select their preferred email server, and then enter their email address and application-specific password. If a user selects multiple email addresses, the messages of the two email addresses can be integrated through the merge step. This implementation of dual-server support is

designed to accommodate the common situation where users may have multiple official email accounts and need to check schedule events from different accounts.

Login information is strictly used to establish a one-time IMAP session and will not be stored locally or transferred to any external service. This design gives priority to privacy and complies with ethical data processing principles.

After submitting the credentials, the user clicks the "Login and Fetch email" button to start the data collection process. Behind the scenes, the system connects to the appropriate IMAP server, retrieves the recent emails within a predefined range from the `INBOX` and `SENTBOX` folders, organizes them into session threads, and saves them in a structured format. This dual-folder method ensures the inclusion of inbound and outbound messages, thereby preserving the complete context of the scheduling discussion (Figure 4.2).

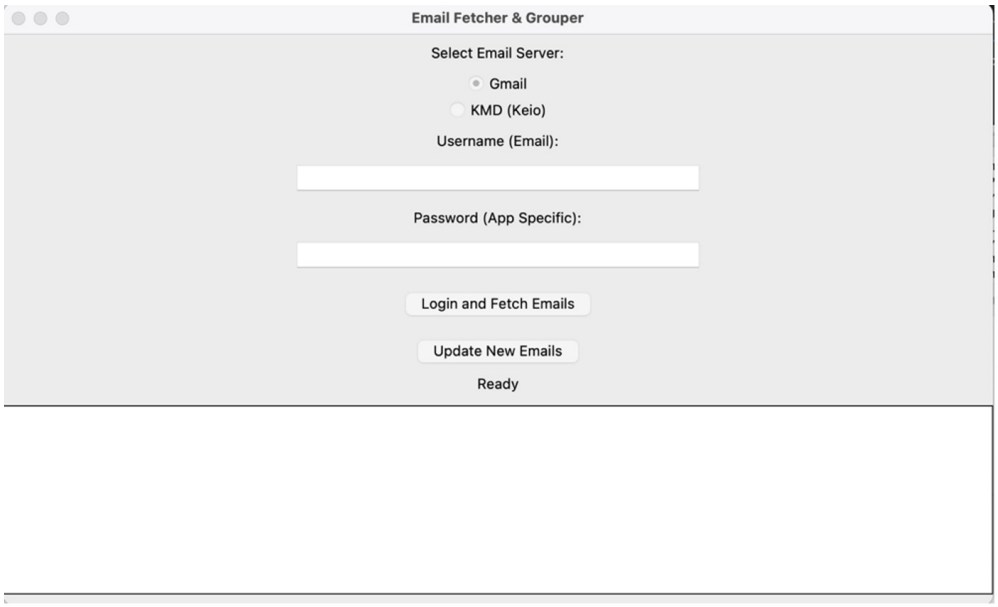

Figure 4.2 GUI of Login

To enhance usability, the interface provides real-time status messages and visual indicators (e.g., progress bars) to inform users of connection status, retrieval progress and completion status. These design options aim to reduce uncertainties in potential technical processes and support non-professional users in successfully setting up the system.

The entire workflow is deliberately kept local: emails are processed in memory and stored in structured JSON format on the user's device. No external uploads or server-based processing will occur. This decision is rooted in the system's privacy-first architecture. This method can not only reduce security risks, but also comply with institutional policies that may prohibit external data sharing.

4.4.2 Visualization of Extracted Events Calendar

After acquisition and analysis, the system displays the results in the GUI calendar interface, showing the schedule of the current week. The calendar uses a weekly view layout from Monday to Sunday. For each day, the extracted events are presented in the form of time blocks with summary details, sender information and extraction sources, as shown in Figure 4.4. The current date is visually highlighted to provide direction. Users can clearly see each event through the calendar interface. To improve orientation and navigation, the current day is visually highlighted, and users are provided with intuitive controls to move between weeks as illustrated in Figure 4.3. Navigation buttons such as "`<Previous Week`" and "`Next Week>`" allow seamless browsing, while consistent layout and minimal visual clutter ensure usability for both technical and non-technical users.

A key design feature is the integration of an LLM prompt input box. This enables users to provide custom instructions (e.g.,

```
Exclude emails from no-reply@slack.com
```

)

which are appended to the prompt by clicking "`Add Prompt`" and sent to the Gemini model during the next round of analysis. This interactive prompt mechanism empowers users to personalize event extraction in real time, without modifying system code or settings. This step allows for the addition of multiple prompts and is displayed on the interface. Multiple prompts can be layered and managed within a single session, and the system retains them in memory for context continuity.

When the user clicks "`Refresh Calendar`", the system reprocesses all previously fetched emails using the updated prompt set to achieve a personalized

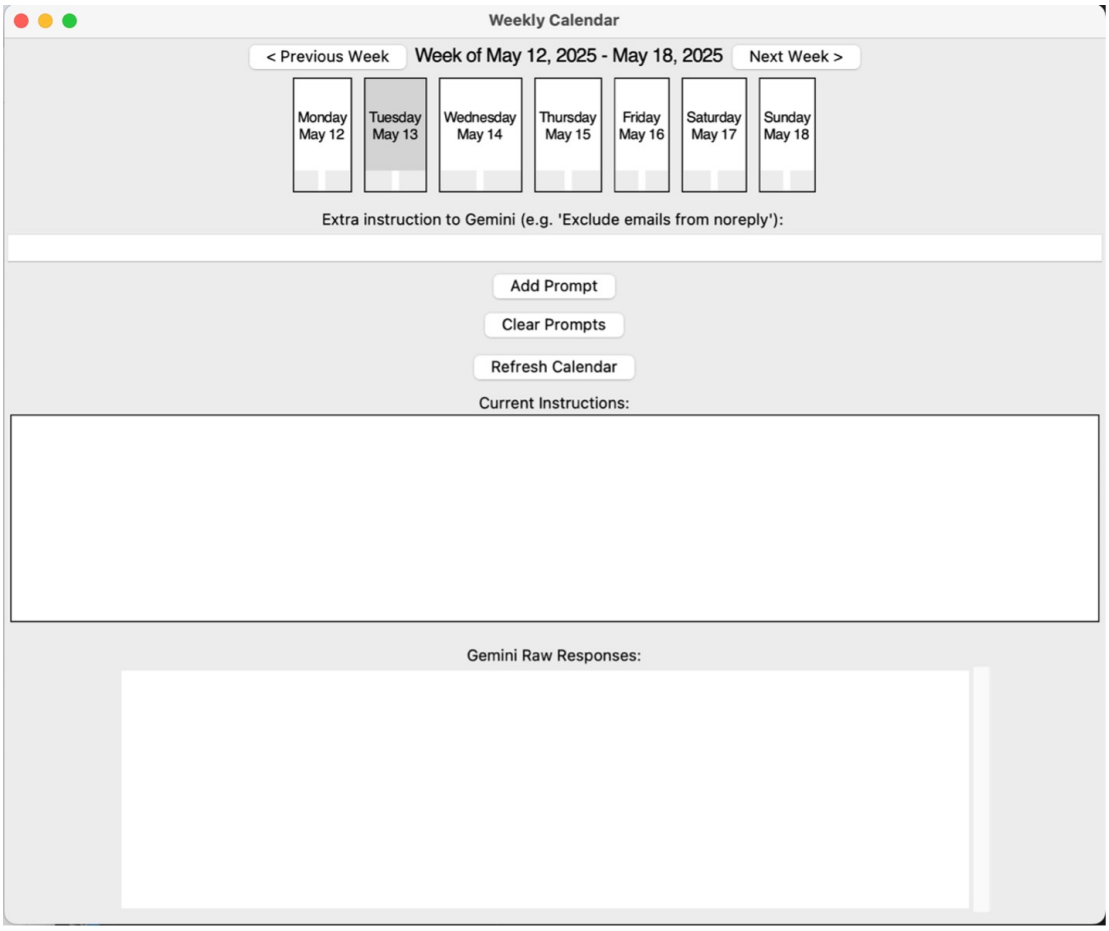

Figure 4.3 GUI of Calendar (Initial View)

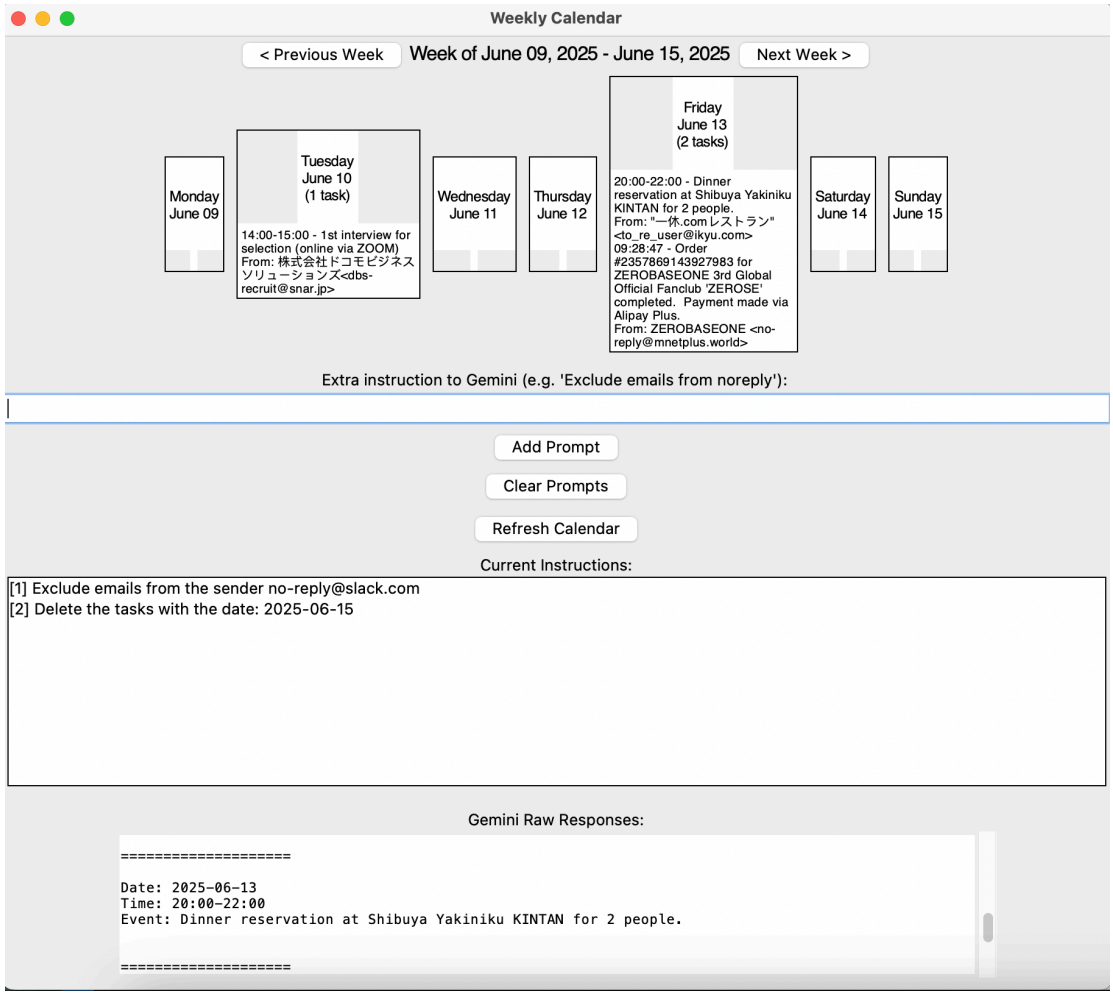

Figure 4.4 GUI of Calendar (After Event Extraction)

function. Meanwhile, Gemini's system output is displayed at the bottom of the interface, allowing users to compare source responses and evaluate quality.

4.5. Update Flow for Real-Time Scheduling

4.5.1 User-Initiated Manual Update Process

This system supports on-demand updates through a manual update mechanism, enabling users to control when to refresh the scheduling information at any time as needed. This enables updates when the user has recently received or sent time-sensitive emails and updates the calendar without restarting the program.

After clicking the "`Update New Mail`" button in the GUI, the system connects to the IMAP server and performs incremental checks based on the unique message identifier(uid)(Figure 4.2). Only obtaining previously unprocessed emails significantly reduces computational overhead and prevents duplicate analysis. Then, the retrieved emails will be delivered through the same thread grouping, event extraction and preference filtering pipeline used in the initial login stage.

To maintain the integrity of past data, the update process only appends: New event entries are added to the calendar interface and the event log file, without modifying or deleting existing entries. This ensures the traceability and reproducibility of historical event data.

This explicit user operation helps maintain the user's perception and control of the system status.

4.5.2 Background Auto-Update Mechanism

In addition to manual updates, the system also includes a background automatic update function, which is specially designed for passive or long-term operation scenarios. This function regularly checks new emails at user-defined intervals (by default every 60 minutes) and performs silent updates without the need for manual updates by the user.

Automatic update is implemented using the thread-based scheduler timing service. Similar to manual updates, this mechanism respects the deletion of duplicate data based on `uid` to avoid reprocessing previously processed messages. It also

follows all the preference filters and prompt instructions provided by the user, maintaining behavioral consistency across update modes.

This feature is particularly important in experimental or production environments because the application is expected to run continuously for several days (e.g., a full week during the user research period). To ensure transparency, each automatic update cycle is recorded, and a summary (e.g., the number of new events added, the updated threads) is selectively displayed or stored in the status file.

This modularization enables users to flexibly choose the active or passive operation mode of the system.

4.6. Privacy Considerations and Data Handling Policy

4.6.1 Local-Only Event Processing and Storage

Privacy is a core design principle of this system. To safeguard user data and ensure compliance with institutional data handling policies, all processing procedures are executed locally on the user's device. This design ensures that no email content, metadata, or extracted information is transmitted to third-party servers or cloud-based processing services at any stage.

Emails are accessed via encrypted IMAP sessions and temporarily stored in system memory during the analysis process. Upon completion, only the essential scheduling information, such as event time, summary, and associated email identifiers, is retained, while all original message content can be manually purged from memory to reinforce privacy protection.

Even when interacting with external APIs (e.g., the Gemini LLM), the system is configured to utilize a secured, paid version of the service and does not retain or transmit any data for the purpose of model training or logging.

Furthermore, when submitting or reviewing the log file, users retain full control over the visibility of personal data. Identifiable information, such as sender names, email addresses, or sensitive message content, may be selectively redacted or abstracted by the user as needed.

This constrained data exposure model significantly reduces the risk of unintended information leakage and ensures the system's suitability for deployment in sensitive or privacy-critical environments.

4.6.2 User Editable Logging Policy

All extracted events are stored locally in a file named `event_email_log.json` located in the same directory as the main program. This log file serves both as a persistent data repository and a human-readable audit trail, capturing the full output of the schedule extraction process in a structured JSON format.

Each log entry contains key fields, including:

- `timestamp`: the original email reception time
- `email_subject`: the subject line of the corresponding email
- `regex_extracted_from_email`: original lines matched by regular expressions
- `gemini_response`: full structured output from the Gemini LLM

The log file is fully editable and designed to give users control over sensitive content prior to submission or further use. Users may open the file using any plain text or JSON-compatible editor.

To protect sensitive information, users are encouraged to manually review the log file and apply redactions where necessary. This may include replacing specific names or phrases with placeholders such as `xxx`, or substituting entire lines or subjects to obscure sensitive content.

To ensure the log file remains structurally valid, users should avoid removing key fields or deleting entire entries. Basic formatting must be preserved to maintain compatibility with the system.

This editable logging mechanism ensures transparency while maintaining the structural integrity required for subsequent processing. It supports privacy data sharing or integration with other dispatching platforms after manual correction or annotation. By granting users full supervision rights over the extracted content, the system complies with ethical data processing principles and the autonomy of

participants in the research and application environment over sensitive private information.

Chapter 5
Implementation

5.1. Project Directory Structure

The implementation of the email-based scheduling system is organized into clearly separated modules, each responsible for a specific functionality such as email fetching, task extraction, thread merging, and user interface rendering. The project follows a modular design to improve maintainability and readability.

All source files are located at the root level, while output data such as processed email threads and extracted task logs are saved in structured JSON format.

The files are classified below by their functional roles:

Source Code Files:

- `fetch_gui.py` -GUI interface: login, fetch, group, display
- `merge_threads.py` -Merges threads
- `regex_schedule.py` -Extracts date/time strings using regex
- `gemini_schedule.py` -Extracts tasks using Gemini LLM
- `calendar_context.py` -GUI interface: calendar and prompt
- `logging_utils.py` -Logs extracted data with duplication check

Data Output Files:

- `emails_grouped_merged.json` -Merged threads
- `parsed_results.json` -Cached Gemini output and parsed tasks
- `event_email_log.json` -Log of extracted events

```
mail = imaplib.IMAP4_SSL(server, port, timeout=60)
mail.login(username, password)
```

Figure 5.1 Login to IMAP Server

5.2. Environment and Tools Used

This system is developed using the Python programming language. It retrieves email data through the IMAP protocol and stores processed threads and extracted events in JSON format. The user interface is implemented with Tkinter, offering a simple and intuitive way to visualize extracted schedules.

5.2.1 IMAP-Based Email Source Management

The system connects to the user's mailbox via the IMAP protocol, allowing retrieval of both inbox and sent emails. Emails are grouped into conversation threads using metadata such as `Message-ID` and `In-Reply-To` headers, enabling context-aware event extraction.

The corresponding Python code to login with the IMAP Server is illustrated in Figure 5.1.

This code initializes a secure SSL connection to the IMAP server and logs in using the user's credentials. It is the foundational step for accessing email folders and performing operations on them.

5.2.2 JSON-Based Thread Storage

To facilitate persistent storage and structured processing of email content, the system stores all grouped email threads and extracted results in JSON format. This approach offers flexibility, human readability, and compatibility with other modules and tools.

The core function `group_emails_by_thread(emails)` maps email conversations into coherent threads by resolving `Message-ID` and `In-Reply-To` relationships. Each thread is stored as a list of email dictionaries, preserving the temporal and logical structure of user communication (Figure 5.2).

```
def group_emails_by_thread(emails):
    id_map = {e.get("message_id"): e for e in emails if e.get("message_id")}
    threads = defaultdict(list)
    for email_data in emails:
        thread_id = email_data.get("in_reply_to") or email_data.get("message_id")
        while thread_id in id_map and id_map[thread_id].get("in_reply_to"):
            thread_id = id_map[thread_id].get("in_reply_to")
        threads[thread_id].append(email_data)
    return list(threads.values())
```

Figure 5.2 Group Emails into Threads

```
def save_to_json(email_data, filename):
    try:
        with open(filename, "w", encoding="utf-8") as f:
            json.dump(email_data, f, ensure_ascii=False, indent=4)
        messagebox.showinfo("Success", f"Emails saved to {filename}")
    except Exception as e:
        messagebox.showerror("Error", f"Error saving emails to JSON: {e}")
```

Figure 5.3 Save Threads to JSON File

Once threads are formed, the function `save_to_json(email_data, filename)` is used to write them to disk in a UTF-8 encoded JSON file, with indentation for easy inspection and debugging (Figure 5.3).

To prevent redundant processing, previously fetched email UIDs are tracked in `seen_uids.json`. These helper functions manage reading and writing that file (Figure 5.4).

During update mode, new email threads are merged into a persistent file (Figure 5.5):

Lastly, a combined file `parsed_results.json` holds the final output of task lists, responses, and threads for reuse in GUI calendar rendering and research evaluation (Figure 5.6).

This JSON-based storage strategy provides transparency, enables post hoc cor-

```
def load_seen_uids():
    if os.path.exists("seen_uids.json"):
        with open("seen_uids.json", "r") as f:
            return set(json.load(f))
    return set()

def save_seen_uids(uids):
    with open("seen_uids.json", "w") as f:
        json.dump(list(uids), f)
```

Figure 5.4 Seen UID Storage and Retrieval

```
with open("emails_grouped_merged.json", "w", encoding="utf-8") as f:
    json.dump(merged_threads, f, indent=2, ensure_ascii=False)
```

Figure 5.5 Merged Threads to JSON

```
result = {
    "task_list": combined_task_list,
    "threads": combined_threads,
    "responses": combined_responses
}
with open("parsed_results.json", "w", encoding="utf-8") as f:
    json.dump(result, f, ensure_ascii=False, indent=2, default=str)
```

Figure 5.6 Parsed Results Save

rection, and ensures data traceability across multiple processing layers.

5.3. Event Extraction Module

This module is responsible for extracting structured schedule events from grouped email threads using two complementary methods: regex-based filtering and LLM-assisted analysis.

5.3.1 Regex-Based Matching Patterns

To filter potential date/time-related sentences from raw email content, a custom regular expression pattern is applied (Figure 5.7). This pattern captures common time formats.

```
1 def extract_date_lines_with_regex(text):
2     pattern = re.compile(
3         r"(\d{1,2}[/-]\d{1,2}|\d{4}[/-]\d{1,2}|\d{1,2}(日|月|号|時|時半)|年\d{1,2}
              月\d{1,2}日|"
4         r"\b\d{4}[-/]\d{1,2}[-/]\d{1,2}|"
5         r"(?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)[a-z]*\s+\d{1,2},\s+\
              d{4}|"
6         r"\d{1,2}\s+(?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)[a-z]*\s+\d
              {4}\b)"
7     )
```

Figure 5.7 Regex-Based Filtering of Date Strings

This helps identify date-related sentences independently of the language model, allowing parallel extraction logic that complements LLM-based analysis.

5.3.2 LLM Prompt Strategy

As discussed in Section 4.2.2, the prompt construction plays a crucial role in guiding the LLM's extraction behavior, and its iterative refinement was integrated into the implementation process.

The optimized system dynamically constructs a multilingual prompt for each email thread, instructing the LLMs to extract structured schedule-related information. This prompt includes logic for:

- Parsing multilingual content
- Interpreting vague or relative time expressions (e.g., “tomorrow”, “next week”) using the email’s received timestamp
- Extraction of only finalized plans, not tentative suggestions
- Inferring implicit events when the email requests actions like “please check the official site” or “see the attachment”
- Handle recurring events (e.g., “every Saturday”) or exceptions (e.g., “except for Dec 14th”)
- Supporting filtering via user-defined custom prompts

This prompt strategy ensures consistent interpretation across multi-lingual threads, improves temporal resolution, and supports both explicit and inferred event cues. For reference, the detailed prompts used during LLM-based extraction can be found in the appendices.

5.4. Graphical User Interface (GUI) Features

5.4.1 Email Fetching via User Login

The system provides a login GUI (Figure 4.2) where users can input their email credentials and initiate IMAP-based fetching.

- **Username/Password Entry Field:** For inputting email login credentials.
- **Login and Fetch Email Button:** Initiates full email fetching from inbox and sent folders across one week.
- **Update New Emails Button:** Fetches only newly arrived or sent emails since the last update and appends results to logs and calendar without overwriting prior data.

Additionally, after the calendar is rendered, the system performs an automatic update every hour in the background. This enables users to passively maintain an up-to-date schedule display, thereby enhancing temporal efficiency.

5.4.2 Visualization of Extracted Events Calendar

The extracted events are visualized in a calendar-style GUI (Figure 4.3) organized by week (Monday-Sunday), with interactivity and filtering options.

- **Weekly Calendar Display:** Shows events grouped by date, with time, task summary, and sender.
- **<Previous Week> Button:** Navigates to the previous week's schedule.
- **Next Week> Button:** Advances to the following week's schedule.
- **Prompt Entry Textbox:** Allows users to enter additional Gemini filtering instructions (e.g., "Exclude emails from noreply").
- **Add Prompt Button:** Adds the entered prompt to the active Gemini instruction set.
- **Clear Prompts Button:** Resets all LLM prompts to default.
- **Refresh Calendar Button:** Reanalyzes all email threads using updated prompt instructions and refreshes the display.
- **Raw Response Viewer:** A scrollable textbox displaying raw Gemini output for transparency and debugging.

This modular and visually structured interface improves user control over event extraction results while maintaining readability and actionable clarity.

5.5. Log File Management and User Operations

5.5.1 Structure and Location of Event Log File

For logging extracted event data, the system appends both regex-based and Gemini-based results to a log file to avoid duplication and ensure traceable storage.

Each log entry includes:

- `timestamp` — Time when the email was received.
- `email_subject` — Subject line of the email.
- `regex_extracted_from_email` — Raw text lines that matched regex patterns.
- `gemini_response` — Parsed schedule information output by the Gemini model.

All extracted scheduling information whether derived via regular expression matching or LLM-based processing is stored locally in a single log file named `event_email_log.json`.

5.5.2 User Editing Procedures

Participants are permitted to make limited manual edits to the log file before submission. Acceptable edits include:

- Replacing names or specific keywords with “xxx”.
- Redacting full sentences with “x” per line while keeping the structure intact.
- Replacing sensitive subject lines with “xxx”.

These constraints ensure that the data format remains machine-readable and structurally valid for post-processing.

Chapter 6
Evaluation

6.1. Evaluation Objectives

The primary goal of this evaluation is to measure the effectiveness and usability of the proposed email-based schedule assistant in real-world scenarios. Specifically, we aim to assess:

- The system's ability to accurately extract scheduling-related information from both received and sent emails.
- The degree to which extracted events match user expectations and actual calendar commitments.
- The impact of interactive features (e.g., additional prompts, manual corrections) on final schedule quality.
- Overall user satisfaction and perceived usefulness after one week of active system use.

Both quantitative (event-level metrics) and qualitative (survey-based) methods are employed to evaluate the assistant's performance across a two-week period.

6.2. Experimental Setup

6.2.1 Participants and Email Sources

Participants were recruited from a graduate student population with diverse email habits. Each participant was instructed to install the tool on their local device. The system supports both institutional (e.g., university) and personal (e.g., Gmail) accounts via IMAP.

Email sources included:

- **Received emails (2 weeks)**: All incoming messages in the inbox during the study period.
- **Sent emails (2 weeks)**: Outgoing messages authored by the user, often critical for capturing agreed schedules.
- **Integrated threads**: Conversations that combine incoming and outgoing messages, grouped using `Message-ID` and `In-Reply-To` fields.

6.2.2 Email Collection and System Execution

The evaluation was divided into two sequential phases:

1. **Initial Deployment Phase (Week 1)**: Upon installation, the system fetched and analyzed one week's worth of emails (received and sent). The extracted event candidates were stored and rendered in the calendar interface.
2. **User Experience Phase (Week 2)**: Users continued their daily use of email. The system automatically updated every hour to fetch newly arrived messages and extract additional events. Users were encouraged to inspect, revise, and validate the extracted events through the GUI. Event log files were updated accordingly.

The system ran entirely locally to preserve privacy. No email data were uploaded to external servers.

6.2.3 Feedback of Event Collection

Throughout the two-week experiment, participants provided systematic feedback on the event extraction process. The evaluation was structured to capture both the system's autonomous extraction capability and the collaborative correction effect enabled through user interaction.

Participants were first presented with a list of **initially extracted events** generated by the system after analyzing their received and sent emails. They were then asked to:

- Mark which extracted events were correctly captured.
- Identify any missed events that were important for their actual schedule.
- Edit or refine events using the calendar interface and prompt re-entry field.

Additionally, the following types of user judgments were collected through interviews and logs:

- **Events extracted after interaction**: Entries confirmed or corrected through prompt-based refinement or calendar editing.
- **Related events (100% must-check)**: Emails that users deemed necessary to read, regardless of whether the system extracted them as events.
- **Core events**: High-impact events where the user must participate or act before a deadline.
- **Error events**: Incorrectly extracted items, including false positives or vague references wrongly identified as confirmed plans.

These classifications helped assess how much of the user's real scheduling intent was covered by the system and how much required human correction.

6.2.4 Log File and Annotation Protocol

As part of the evaluation design, the system was configured to automatically generate a structured log file, `event_email_log.json`, during the experimental usage period. This file served as the primary record of extracted scheduling information, combining both regular expression-based findings and Gemini LLM outputs. The log was stored locally on the participant's computer to ensure data privacy and facilitate direct inspection.

To support the creation of a ground truth dataset for objective evaluation, participants were asked to review the content of this log file after one week of use. They were permitted to manually annotate or correct the entries based on their understanding of the emails. To preserve structural integrity while protecting sensitive content, anonymization instructions were provided: replacing proper nouns

```
{
  "timestamp": "2025-06-05T12:55:43+00:00",
  "email_subject": "【xxx】ご予約完了のお知らせ",
  "regex_extracted_from_email": [
    "x",
    "2025/06/06(金) 19:00",
    "2時間"
  ],
  "gemini_response": "Date: 2025-06-06\nTime: 19:00-21:00\nEvent: xxx\n"
},
```


Figure 6.1 Example of User-Annotated Event Log Showing Modifications

or identifiers with "xxx", masking specific lines with "x" such as Figure 6.1, and avoiding removal of fields.

Any perceived extraction errors, such as missing, incorrect, or irrelevant events, could be identified during post-task interviews and verbally reported to the research team. This interview-based feedback, in combination with the annotated log file, enabled the construction of a reliable ground truth dataset against which qualitative interpretations of the system's performance could be evaluated.

6.2.5 User Survey for Subjective Feedback

A multilingual questionnaire was distributed at the end of the experimental period to collect participants' subjective impressions. The survey was structured into four sections:

- **Section A: User Habits**
  Captured user behavior regarding email checking frequency, scheduling methods, and current calendar usage.
- **Section B: Perceived Accuracy**
  Asked users to estimate the percentage of correctly extracted events, report missed or irrelevant ones, and evaluate accuracy after manual prompt interaction.
- **Section C: Before-and-After Comparison**
  Measured impressions on system performance before and after editing or reviewing results, and gauged preference for manual verification.

- **Section D: Attitudes Toward System Use**
  Collected perspectives on the usefulness of the email-based scheduling system in various real-life contexts (e.g., busy weeks, travel periods), and included a 1-10 scale for overall satisfaction.

This structured feedback helped assess not only the technical performance of the system, but also the practical value and user experience, forming a basis for further improvements in future iterations.

6.3. Quantitative Evaluation

6.3.1 Initial Extraction Accuracy

Based on the analysis of log file data from 12 participants over a two-week period in the experiment, as shown in Table 6.1, each user received an average of 95.6 emails, sent 2.3 emails, and formed 91.1 integrated threads, and the system-level approach demonstrates consistently high extraction performance across users, with an average of 69.7 events extracted per user. In comparison, the pattern-based method yields a lower average of 60.5 events. The number of extracted events increased for nearly all users. This suggests that the system can capture a wider range of scheduling information from email threads by integrating rule-based patterns with Large Language Models (LLMs) inference.

Discussion:

The system demonstrates a clear advantage in inferring implicit scheduling intents, especially when emails contain indirect references such as “please check the official site” or “see the attachment for the schedule.” These expressions lack explicit time or date information and were often ignored by the pattern-based extractor. In contrast, the system leverages contextual understanding to interpret these instructions as valid event triggers, enriching the extracted schedule.

Furthermore, the system performs better in handling emails where schedule information is embedded in hyperlinks or attachments - common in service reservations, travel plans, and entertainment bookings (e.g., Disneyland reservations

Table 6.1 Email Activity and Comparison of Extracted Events by Pattern and System

User ID	Received Email (Two Weeks)	Sent Email (Two Weeks)	Integrated Thread	Pattern-Based Extracted Events	System-Based Extracted Events
1	115	2	109	45	50
2	90	0	89	35	35
3	68	0	68	23	27
4	100	2	99	63	92
5	59	1	58	51	59
6	73	0	71	45	53
7	67	0	65	41	42
8	125	1	118	99	138
9	123	1	123	106	115
10	110	8	102	90	98
11	87	11	68	60	68
12	130	2	123	58	59

(Figure 6.2)). Such formats often follow templates that lack standard temporal expressions and instead direct users to external content. Pattern-based approaches tend to underperform in these scenarios due to their reliance on in-body regular expressions, whereas the system approach demonstrates superior flexibility and semantic reasoning.

These findings highlight the necessity of incorporating language models capable of contextual understanding to support real-world scheduling needs from diverse and loosely structured email formats.

6.3.2 Accuracy Improvements After Interaction

We conducted a comparative analysis of the three data groups using box plots to examine the system improvement rate and the manual improvement rates of related events and core events as human baseline references. The boxplots illustrate the distribution of improvement rates across participants, highlighting median values, interquartile ranges, and outliers.

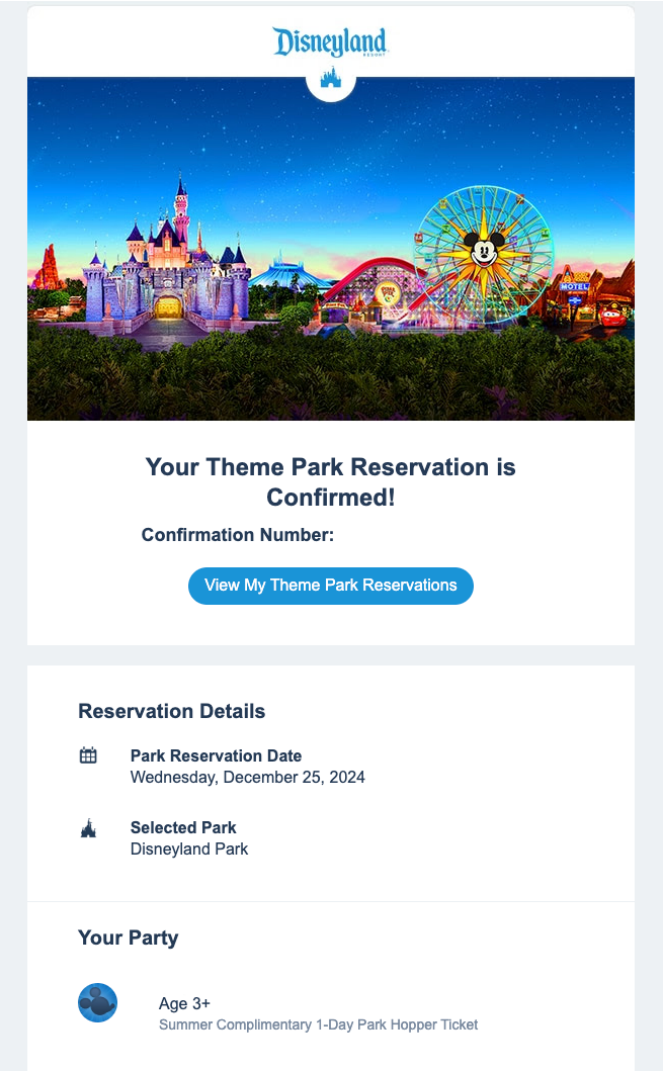

Figure 6.2 Hyperlinks Example of Reservations for Disneyland

In this context, **core event** refers to the schedule items that require actual user operation or attendance, such as meetings with deadlines or confirmed appointments. On the contrary, **related events** contains supplementary information fragments that users may need to be aware of in addition to the core events (e.g., event announcements, invitations or notifications), but it does not necessarily imply specific commitments. These categories were manually marked during the evaluation period to distinguish between necessary and unnecessary scheduling contents.

Let E_{initial} be the number of events initially extracted, and E_{after} be the number of events confirmed or retained after user operation (either by the user interaction with system or manual operation by the user (e.g., correction, deletion, or confirmation)). We define three specific improvement rate indicators as follows:

- **System Improvement Rate (Related Event):**

$$\text{System Improvement Rate}_{\text{related}} = \frac{E_{\text{initial}}^{\text{system}} - E_{\text{after}}^{\text{interact}}}{E_{\text{initial}}^{\text{system}}} \tag{6.1}$$

  This reflects the degree to which system-extracted related events were adjusted or rejected by the user interaction with the system.

- **Manual Improvement Rate (Related Event):**

$$\text{Manual Improvement Rate}_{\text{related}} = \frac{E_{\text{initial}}^{\text{system}} - E_{\text{after}}^{\text{manual, related}}}{E_{\text{initial}}^{\text{system}}} \tag{6.2}$$

  This evaluates how much the manually extracted related events required revision compared to those extracted systematically, serving as a human baseline.

- **Manual Improvement Rate (Core Event):**

$$\text{Manual Improvement Rate}_{\text{core}} = \frac{E_{\text{initial}}^{\text{system}} - E_{\text{after}}^{\text{manual, core}}}{E_{\text{initial}}^{\text{system}}} \tag{6.3}$$

  This reflects the rate of modification for manually extracted core events after further checking compared to those extracted systematically, serving as a human baseline.

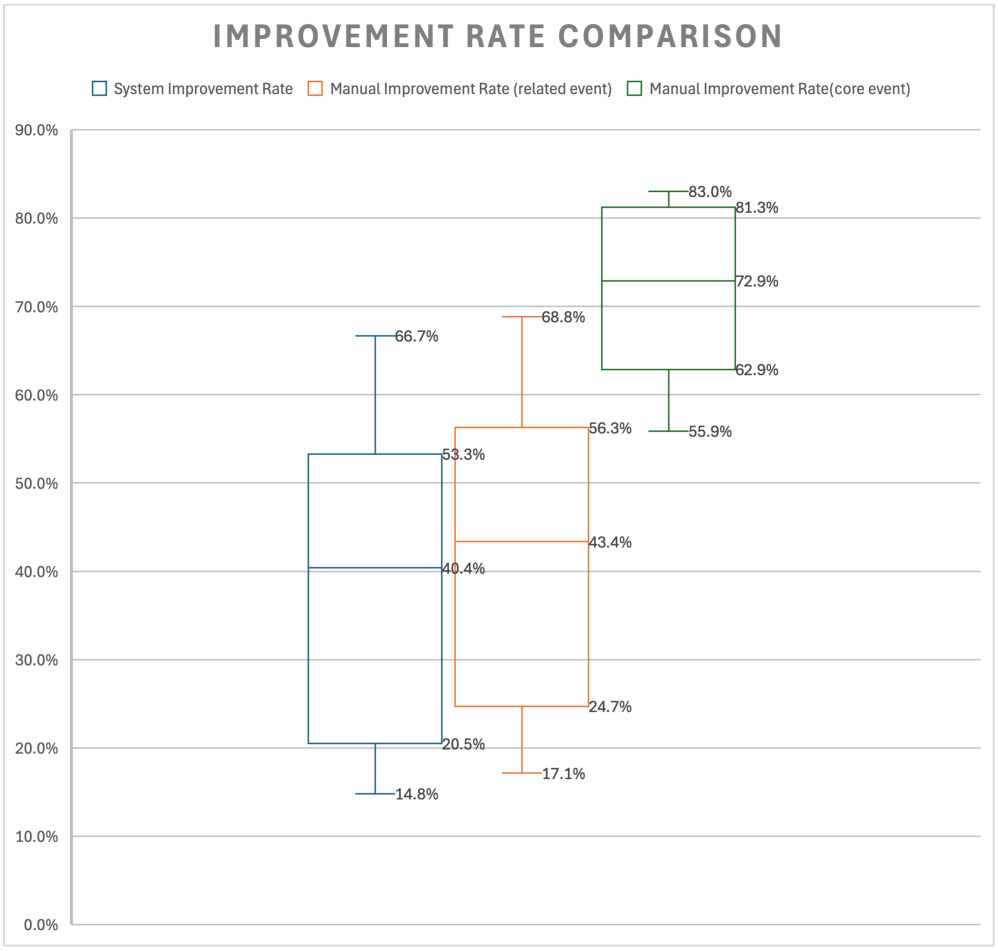

Figure 6.3 Event Extraction Improvement Rate Comparison between the System and Manual

As shown in Figure 6.3, the improvement rate reflects how much the initially extracted events were revised or removed by the user. A lower improvement rate indicates higher initial accuracy and reduced need for user correction.

Among the three compared groups, the **system improvement rate for related events** shows a relatively low median of 40.4%, indicating that the system's initial extraction of related events already achieves a reasonable level of precision. After users interact with the system through prompt-based refinement, the **manual improvement rate for related events** is only slightly higher at 43.4%. The similarity between these two distributions suggests that the system performs comparably to human annotators in the extraction of related events.

In contrast, the **manual improvement rate for core events** is significantly higher, with a median of 72.9%. This indicates that users made substantial revisions to manually extracted core events, likely due to over-inclusion or difficulty in distinguishing essential commitments and deadlines from general contextual information.

In summary, the results demonstrate that the system has reached a practically usable level for related event extraction, where user interaction mainly serves to make minor refinements. However, for core event recognition, the system still lacks the precision and contextual understanding needed for autonomous extraction. Future improvements should focus on enhancing the system's ability to identify essential, action-relevant information in diverse and often implicit email contexts.

Discussion:

Although the results show that the system performs relatively well in extracting relevant events — its performance can be close to manual extraction by humans, the extraction of core events remains a significant challenge. One influencing factor is the inherent limitation of relying on simple user prompts to guide Large Language Models (LLMs). Users may have inconsistent or ambiguous wording prompts, and the current prompt design often fails to capture the implicit scheduling intentions embedded in complex or multi-layered email content.

Furthermore, even if LLMs receive effective prompts, they still have limitations in terms of temporal reasoning, selective filtering, and priority ranking. For example, events that distinguish information content from action requirements

(e.g., deadlines or confirmed meetings) are usually not fully modeled by LLMs and cannot be precise to such events by enhancing accuracy.

Moreover, our observations suggest a clear asymmetry in prompt effectiveness: prompts aimed at **excluding** specific content, such as "ignore schedules from `xxx`" or "remove the one on [date]", are interpreted relatively reliably and consistently by the model. In contrast, prompts that attempt to **prioritize or selectively retain** content, such as "only include emails from `xxx`" or "focus on confirmed schedules only", tend to lead to significant misclassifications. In many cases, such prompts result in the unintentional removal of necessary events or inclusion of unrelated information.

These findings underscore the need for continued advancements in LLM capabilities, particularly in handling core event extraction and prompt-based prioritization.

6.3.3 False Positives and Missing Events

To assess the precision and coverage of system-extracted events, we analyzed the relationship between the number of initially extracted events and manually evaluated error events, including both false positives and missing items. The error rate was calculated by dividing the number of error events by the total number of initially extracted events for each user.

Table 6.2 summarizes the evaluation results across 12 participants. In this table, **positive integers** in the "error event" column represent **false positives** — unnecessary or redundant events mistakenly extracted by the system. In contrast, **negative integers** indicate **missing events** — important scheduling items that were not captured.

Among the 12 participants, error rates ranged from **-3.1%** to **7.4%**, with most values falling within the 0% to 5% range. In several cases (Participants 2 and 4), the system achieved a 0.0% error rate, suggesting high precision for emails with explicit dates.

Table 6.2 Error Event Evaluation per User

User ID	Initially Extracted Events (sys 100%)	Error Event	Error Rate
1	50	2	4.0%
2	35	0	0.0%
3	27	2	7.4%
4	92	0	0.0%
5	59	3	5.1%
6	53	2	3.8%
7	42	1	2.4%
8	138	3	2.2%
9	115	-1	-0.9%
10	98	-3	-3.1%
11	68	1	1.5%
12	59	1	1.7%

Discussion:

A qualitative analysis of user feedback revealed several distinct categories of system errors in schedule extraction. These highlight limitations in both language understanding and contextual reasoning. The main error types are summarized as follows:

1. **Time Zone Misalignment**: The issue observed involved global time zone differences. In certain threads, events were scheduled with expressions like "June 3rd" based on the sender's timezone, but the system interpreted them as "June 2nd" based on the receiver's local time. This led to incorrect date extraction due to a lack of awareness of time zone context in email headers.

2. **Final Time Misidentification in Repeated Negotiation**: If absolute dates are consistently used throughout the scheduling conversation, the system can correctly identify the final confirmed time. However, when relative time expressions such as "tomorrow" and "Wednesday" are repeatedly used during negotiation, the system fails to determine which one was ultimately agreed upon. In one case, both "tomorrow" and "Wednesday" were extracted, even though only "Wednesday" was the confirmed date.

3. **Duplicate Extraction from Multi-Threaded Cancellations**: When a user canceled a reservation via a web interface, confirmation was sent in multiple system-generated emails, each with a distinct Message-ID. The system failed to recognize that these referred to the same canceled event, and mistakenly recorded them as separate valid events due to the lack of cross-thread consolidation.

4. **Misinterpretation of Relative Time Expressions**: The system struggled with vague or relative time phrases like "as soon as possible" or "next week," especially when the reference point (e.g., email timestamp) was not correctly anchored, leading to ambiguous or misaligned scheduling.

5. **Extraction of Non-Attended or Informational Events**: Some events were added to the calendar simply because a date was mentioned in an email, even when the user had not committed to participation, such as concert announcements before lottery results. This highlights the need for clearer intent recognition.

6. **Date Parsing Errors from Non-Date Tokens**: In several cases, numeric strings such as reservation IDs (e.g., "#123456") were misinterpreted as calendar dates, resulting in invalid entries.

7. **Unrecognized Cancellation Deadlines**: Temporal rules like "this can be canceled two days before the event" were not extracted or treated as meaningful scheduling constraints, despite users explicitly valuing this information.

8. **Omission of Flexible User-Scheduled Tasks**: Tasks with no fixed deadline, such as "asap" items determined by the user, were overlooked by the system, revealing a gap in its ability to capture informal or user-defined scheduling logic.

In summary, the system demonstrates weaknesses in temporal resolution, event consolidation, and semantic intent classification. Particular attention should be given to improving the interpretation of relative time and cancellation semantics, as well as expanding the model's ability to capture user-defined, flexible event types.

6.4. Qualitative Evaluation

6.4.1 User Questionnaire Results

To evaluate the perceived usefulness, usability, and user acceptance of the email-based schedule extraction system, we conducted a qualitative user survey involving 12 participants. Participants completed a structured questionnaire after interacting with the prototype system, and their responses were categorized into four thematic sections: user habits, accuracy perception, interaction impact, and overall attitude toward the system concept.

Section A: User Habits

All participants (12/12) reported using two or more types of email accounts (e.g., Gmail, university mail, work mail), and the majority (9/12) checked their email multiple times per day. Moreover, all users stated they regularly use email to coordinate schedules. These results suggest that emails are a central part of participants' scheduling workflows and support the system's premise of email-based schedule extraction.

Section B: Perceived Accuracy

Most users (11/12) felt that the system successfully extracted between 90%-100% of their actual schedule events, indicating a high level of satisfaction with the extraction performance. However, 9 out of 12 participants noted that some irrelevant or incorrect events were also extracted. Follow-up responses revealed common misidentifications such as interpreting phone numbers or reservation codes as dates and extracting vague task-related content without concrete time references. These findings highlight both the system's strong accuracy and its current limitations in semantic disambiguation.

Section C: Before-and-After Interaction

Interactive validation through the calendar interface significantly improved user perception of accuracy. Specifically, (11/12) participants stated that the extraction results became more accurate after interacting with the interface. All partic-

ipants (12/12) preferred having an interaction step before confirming schedules. These insights support the importance of incorporating a human-in-the-loop approach that enables users to verify and correct extracted events, thereby improving trust and system reliability.

Section D: Attitudes Toward System Use

The system concept was well-received overall: 10 out of 12 participants indicated they would consider using the system if released as a product. The average satisfaction score across participants was 7.5 out of 10. Users found the idea of automatically extracting events from emails appealing, particularly when applied to use cases such as receiving many meeting requests, handling travel or reservation-related messages, and coordinating complex schedule negotiations. All participants expressed a preference for an interactive confirmation process rather than relying solely on automatic scheduling.

Discussion:

Overall, the qualitative evaluation demonstrates a high degree of alignment between user needs and system capabilities. The system was regarded as accurate, intuitive, and helpful for real-world scheduling scenarios. Participant feedback also suggested future improvements, including personalized tagging, better handling of vague tasks, and support for financial or to-do event reminders.

6.4.2 User Feedback Summary

Participants also provided open-ended feedback on potential improvements to the system. These responses were categorized into the following themes:

1. **Temporal Understanding and Smart Scheduling**
   (a) Better handling of time zone differences.
   (b) Improved recognition of ASAP-type tasks without fixed deadlines.
2. **Interface and Interaction Enhancements**
   (a) Desire for prioritization suggestions based on negotiation context.

(b) Highlight canceled events more clearly (e.g., red labels).

(c) Highlighter or marker tool for emphasizing important events.

(d) Option to sort events by user-defined importance.

(e) Sound or vibration notifications upon new schedule detection.

(f) Mismatched emails to trigger alerts automatically.

3. **Event Classification and Personalization**

   (a) Request for automatic schedule classification (e.g., work vs. personal).

   (b) Support for user-defined tags and more granular categorization.

   (c) Smart reminders such as large expense detection from bank-related emails.

4. **Usability and Deployment Preferences**

   (a) Preference for lightweight integration, such as a browser extension instead of a separate app.

These suggestions highlight user demand for both deeper semantic understanding and enhanced personalization in future system iterations.

6.5. Answers to the Research Questions

This section presents how each of the four core research questions was addressed in this study, highlighting both the effectiveness of the proposed system and remaining challenges.

6.5.1 Accurate Extraction

The system demonstrates high accuracy in extracting schedule-related events from emails, correctly identifying over **92.6%** of relevant events. As discussed in Section 6.3.3, the error rates across different categories ranged from -3.1% to 7.4%, further supporting the accuracy of the extraction performance. And It successfully handles absolute and relative time expressions, and can process diverse and loosely

formatted email structures. These results indicate that the system is capable of robust time information recognition across real-world scenarios.

However, some challenges remain. The system may fail to detect vague or implicit temporal expressions (e.g., “ASAP”, “check it later”) and sometimes includes unsure events that do not correspond to actual tasks. These limitations suggest room for improvement in handling ambiguous or weakly stated scheduling cues.

6.5.2 Negotiation-Aware Scheduling

This system is capable of identifying the agreed-upon time when explicit time expressions (e.g., “July 21, 2025”) are used in emails within the same thread, thereby enabling effective integration of schedule arrangements during ongoing negotiations.

Nonetheless, the system struggles when dealing with agreements spanning multiple threads, and it is challenging to determine the final confirmed time when using a lot of relative expressions (e.g., “the day after tomorrow”, “next Wednesday”). As noted in Section 6.3.3, the failure cases occurred when the final agreed time was expressed only relatively across threads. These issues highlight the complexity of multi-turn and cross-thread negotiation interpretation.

6.5.3 Cross-Platform Email Integration

The proposed system effectively integrates with major email platforms, including Gmail and university-hosted accounts, via the **IMAP** protocol. This enables unified access to both inbox and sent messages across providers.

However, the system does not yet support other protocols such as POP3 and has not been extended to platforms beyond email. This limits its applicability in contexts where scheduling discussions take place on alternative communication tools.

6.5.4 Personalized and Interactive Filtering

The system provides a **GUI-based interface** (Figure 4.3) that supports LLM-powered prompt refinement and manual editing of extracted events. This inter-

active design significantly improves the perceived accuracy of users, as supported by findings in Section 6.4.1, where user survey results indicated that the ability to manually edit and confirm events contributed to higher satisfaction and trust in the extracted schedules.

However, prompts requiring deep semantic understanding often fail, and there is no built-in mechanism for user preference classification or message importance highlighting. Future enhancements should incorporate adaptive filtering mechanisms and enhance the accuracy of LLMs semantic understanding to better align with individual user needs.

Chapter 7
Conclusion

7.1. Conclusion

In this study, we proposed an email-based scheduling assistant that automatically retrieves emails via IMAP, groups them by thread, and extracts confirmed events using a combination of regular expressions and Large Language Models (LLMs). The system provides a graphical interface for visualizing results and allows users to review and edit logs, ensuring both usability and privacy.

Through real-world evaluation, the system achieved over 90% success in extracting meaningful events without overwhelming users with false positives. By supporting multiple languages and recognizing both absolute and relative temporal expressions, it demonstrates practical utility in informal scheduling scenarios.

7.2. Future Work

Although this study has demonstrated the feasibility and effectiveness of combining rule-based filtering and language model reasoning to assist in email scheduling, there are still several directions that need further exploration and system enhancement.

First of all, although the current prototype only uses the IMAP protocol to retrieve email data, future iterations may include support for other retrieval methods, such as POP3 [16]. This will enhance compatibility with different user environments and increase the robustness of the system.

Secondly, the scheduling extraction pipeline currently mainly focuses on plain text and emails body content. To further expand coverage, especially in business environments, future efforts should explore attachment resolution (e.g., `.pdf`) and deeper integration with common email clients or APIs.

Thirdly, the assessment revealed several minor mistakes in the interpretation of time. In particular, due to the inconsistency of time zones, when the sender and the receiver are in different regions, the date changes. In the negotiation, relative terms such as “tomorrow” and “next Wednesday” are repeatedly used, resulting in confusion in the final determination of the date. Future work should combine the use of time-aware parsing for email headers and enhance time resolution through context disambiguation, especially in negotiation threads.

Fourthly, although user interaction has been proven to be effective in improving extraction accuracy, it still brings cognitive burdens — especially in scenarios that require prioritizing and classification. Users usually need to carefully examine the lengthy output, redefine the prompts, and manually correct errors or irrelevant events. To alleviate this situation, future research should explore more intuitive and effective interaction mechanisms, such as guiding prompt templates, or editing interfaces based on visual highlighting, that reduce manual operations. In addition, improving the underlying performance of LLMs particularly in handling vague user prompts, and understanding nuanced scheduling expressions will be essential for enhancing automation and reducing reliance on user intervention.

Finally, future improvements will be based on user feedback, the system should better support interactive correction without overloading users cognitively. While user intervention has been shown to improve extraction accuracy, it often introduces mental burden — especially when users must prioritize, classify, or refine system outputs. Participants noted the difficulty of reviewing long event lists, redefining prompts, or manually correcting irrelevant entries. To address this, future work should investigate more intuitive and lightweight interaction mechanisms, such as guided prompt templates or visual editing interfaces with highlight-based corrections, which can minimize manual effort. Moreover, enhancing the LLM’s ability to interpret vague prompts and nuanced scheduling expressions is essential to further automate the process and reduce dependence on user intervention.

References

[1] Kostadin Kushlev and Elizabeth W Dunn. Checking email less frequently reduces stress. *Computers in Human Behavior*, 43:220–228, 2015.

[2] Catherine Grevet, David Choi, Debra Kumar, and Eric Gilbert. Overload is overloaded: email in the age of gmail. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, page 793â 802, New York, NY, USA, 2014. Association for Computing Machinery. URL: `https://doi.org/10.1145/2556288.2557013`, `doi:10.1145/2556288.2557013`.

[3] Pauline M Berry, Melinda Gervasio, Bart Peintner, and Neil Yorke-Smith. Ptime: Personalized assistance for calendaring. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(4):1–22, 2011.

[4] Shereen Albitar, Sébastien Fournier, and Bernard Espinasse. An effective tf/idf-based text-to-text semantic similarity measure for text classification. In Boualem Benatallah, Azer Bestavros, Yannis Manolopoulos, Athena Vakali, and Yanchun Zhang, editors, *Web Information Systems Engineering – WISE 2014*, pages 105–114, Cham, 2014. Springer International Publishing.

[5] Maryam Azam, Shah Khalid, Sulaiman Almutairi, Hasan Ali Khattak, Abdallah Namoun, Amjad Ali, and Hafiz Syed Muhammad Bilal. Current trends and advances in extractive text summarization: A comprehensive review. *IEEE Access*, 13:28150–28166, 2025. `doi:10.1109/ACCESS.2025.3538886`.

[6] Bernard Desruisseaux. Internet Calendaring and Scheduling Core Object Specification (iCalendar). RFC 5545, September 2009. URL: `https://www.rfc-editor.org/info/rfc5545`, `doi:10.17487/RFC5545`.

[7] Google Developers. Amp for email, 2024. Accessed: 2025-07-25. URL: `https://developers.google.com/workspace/gmail/ampemail`.

[8] Google Workspace Support. Google workspace smart features for your users, 2025. Accessed: 2025-07-26. URL: https://support.google.com/a/answer/10095404?hl=en.

[9] Reza Yousefi Maragheh, Chenhao Fang, Charan Chand Irugu, Parth Parikh, Jason Cho, Jianpeng Xu, Saranyan Sukumar, Malay Patel, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Llm-take: Theme-aware keyword extraction using large language models. In *2023 IEEE International Conference on Big Data (BigData)*, pages 4318–4324, 2023. doi:10.1109/BigData59044.2023.10386476.

[10] Richard Socher, Cliff Chiung-Yu Lin, Andrew Y Ng, and Christopher D Manning. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 129–136, 2011.

[11] Bryan Wang, Gang Li, and Yang Li. Enabling conversational interaction with mobile ui using large language models. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–17, 2023.

[12] Alexey Melnikov and Barry Leiba. Internet Message Access Protocol (IMAP) - Version 4rev2. RFC 9051, August 2021. URL: https://www.rfc-editor.org/info/rfc9051, doi:10.17487/RFC9051.

[13] Pete Resnick. Internet Message Format. RFC 2822, April 2001. URL: https://www.rfc-editor.org/info/rfc2822, doi:10.17487/RFC2822.

[14] B.M. Gross. *The Managing of Organizations: The Administrative Struggle*. Number 2 in The Managing of Organizations. Free Press of Glencoe, 1964. URL: https://books.google.co.jp/books?id=8IiFAAAAMAAJ.

[15] Samuel R. Bowman. Eight things to know about large language models, 2023. URL: https://arxiv.org/abs/2304.00612, arXiv:2304.00612.

[16] John G. Myers and Dr. Marshall T. Rose. Post Office Protocol - Version 3. RFC 1725, November 1994. URL: https://www.rfc-editor.org/info/rfc1725, doi:10.17487/RFC1725.

[17] Google AI. Gemini api documentation, 2024. Accessed: 2025-07-26. URL: https://ai.google.dev/gemini-api/docs.

Appendices

A. Event Extraction with Gemini Model

The following script illustrates the Gemini-based schedule extraction.

```
GOOGLE_API_KEY = ""  # Google API
os.environ["GOOGLE_API_KEY"] = GOOGLE_API_KEY
llm = Gemini(model="models/(gemini-models)", temperature=0)
    json_file_path = "emails_grouped_merged.json"
user_prompt_history = []
user_prompt_mode = "append"
def extract_tasks_with_gemini(threads, user_prompt_history=None):
    if user_prompt_history is None:
        user_prompt_history = []
    task_list = []
    responses = []
    for idx, thread in enumerate(threads):
        full_conversation = "\n\n".join(
            [f"Subject: {email['subject']}\nFrom: {email['from']}\n{email['body']}"
            for email in thread]
        )
        safe_prompt_lines = [p for p in user_prompt_history if isinstance(p, str)]
        user_extra_prompt = "\n".join(safe_prompt_lines)
        received_time = thread[-1].get("received_time") if thread else None

        prompt = (
        f"Analyze the following email and extract events.\n"
        f"For each event, provide the final confirmed date and time.\n"
        f"Analyze the following email content \n"
        f"and extract all events with:"
        f"1. Date of the event.\n"
        f"2. Time of the event or approximate time range.\n"
        f"3. A brief description of the event.\n"
        f"The email may include English, Japanese, or Chinese.\n\n"
        f"Include Japanese formats like '振込日：2024年12月5日（木）'\n"
        f"and extract both the date and description.\n\n"
        f"Handle recurring events (e.g., every Saturday).\n"
        f"Adjust vague references like 'tomorrow', 'next week'\n"
        f"based on the email's date.\n"
        f"Assume today's date is {received_time}.\n"
        f"Interpret vague expressions using this date.\n"
```

```
        f"Include time ranges like '9am to 11am'.\n"
        f"Handle exceptions like 'except for Dec 14th'.\n\n"
        f"Analyze the thread and extract agreed-upon events.\n"
        f"For each confirmed event, provide:\n"
        f"1. Final date(s).\n"
        f"2. Final time or time range.\n"
        f"3. Concise description.\n"
        f"Only output plans that have been finalized.\n"
        f"Handle vague terms like 'Tuesday', '10am to noon'.\n"
        f"For 'tomorrow' or 'next week', give inferred date.\n\n"
        f"Return multiple entries if needed.\n"
        f"Output format:\n"
        f"Date: YYYY-MM-DD[, YYYY-MM-DD, ...]\n"
        f"Time: HH:MM-HH:MM or similar\n"
        f"Event: short description\n---\n\n"
        f"[Instruction to model: only apply the following filtering "
        f"logic to emails matching the criteria. "
        f"Do not change how you interpret unrelated threads.]\n"
        f"{user_extra_prompt}\n\n"
        f"Thread content:\n\n{full_conversation}"
    )


        try:
            print(f"[Gemini] Processing thread {idx + 1}/{len(threads)}")
            response = llm.complete(prompt)
            response_text = response.text
            email_subject = thread[-1].get("subject", "No Subject")
            received_time = thread[-1].get("received_time") if thread else None
            print("[DEBUG] received_time =", received_time, type(received_time))
            log_event_email(full_conversation, response_text, email_subject,received_time)
            responses.append(response_text)
            matches = re.findall(
                r"Date:\s*(.+?)\s*Time:\s*(.+?)(?:\s*Event:\s*(.+?))?(?=\nDate:|\Z)",
                response_text,
                re.DOTALL
            )
            for date_text, time_text, event_text in matches:
                date_text = date_text.strip()
                time_text = time_text.strip()
                if "removed" in time_text.lower():
                    print(f"[Skip] Skipping removed event: {date_text} - {time_text}")
                    continue
                if event_text:
                    event = event_text.strip()
                elif " - " in time_text:
                    time_text, event = map(str.strip, time_text.split(" - ", 1))
                else:
                    event = "Unspecified"
                date_candidates = re.split(r",|and|&", date_text)
                print(f"[Split Dates] Raw: {date_text} → {date_candidates}")
```

```
                for date_str in date_candidates:
                    cleaned_date = date_str.strip()
                    parsed_date = parse_date(cleaned_date, languages=['en', 'ja', 'zh'])
                    if parsed_date:
                        print(f"→ Parsed task: {parsed_date.date()} - {time_text} - {event}")
                        task_list.append({
                            "date": parsed_date,
                            "time": time_text,
                            "task": event,
                            "from": thread[-1]["from"] if thread else "Unknown"
                        })
                    else:
                        print(f"Could not parse date: {cleaned_date}")
        except Exception as e:
            print("[Gemini Error]", e)
            traceback.print_exc()
        time.sleep(20)
    return task_list, responses
def schedule_logic(prompt_history=None):
    if prompt_history is None:
        prompt_history = []
    with open("emails_grouped_merged.json", "r", encoding="utf-8") as f:
        threads = json.load(f)
    task_list, responses = extract_tasks_with_gemini(threads, prompt_history)
    for task in task_list:
        print(f"→ {task['date'].date()} - {task['time']} - {task['task']}")
    print(f"Total confirmed tasks: {len(task_list)}")
    result = {
        "task_list": task_list,
        "threads": threads,
        "responses": responses
    }
    with open("parsed_results.json", "w", encoding="utf-8") as f:
        json.dump(result, f, ensure_ascii=False, indent=2, default=str)
    return task_list, threads, responses
def main():
    print(f"Loading emails from {json_file_path}...")
    with open(json_file_path, "r", encoding="utf-8") as file:
        threads = json.load(file)
        print(f"Email threads loaded: {len(threads)} groups")
    schedule_logic()
if __name__ == "__main__":
    main()
```