Intelligent Email Assistant: An AI-Powered Email Management System

Lin Yang
University of Illinois Urbana-Champaign
Champaign, United States
linyang6@illinois.edu

Ruoqian Huang
University of Illinois Urbana-Champaign
Champaign, United States
ruoqian5@illinois.edu

Longsheng Yan
University of Illinois Urbana-Champaign
Champaign, United States
lyan11@illinois.edu

Ye Yu
University of Illinois Urbana-Champaign
Champaign, United States
yeyu4@illinois.edu

Abstract

This paper introduces the *Intelligent Email Assistant*, a Chrome extension designed to streamline email management and enhance user productivity. Powered by OpenAI's state-of-the-art language models and integrated seamlessly with Gmail via the Gmail API, the assistant offers three core capabilities: automated email classification, robust spam detection, and natural language query support. It enables users to identify relevant emails, surface potential spam, and interact with their inbox using simple natural language questions. Designed with practicality in mind, the system provides a seamless and intuitive experience directly within the Gmail interface, making intelligent email management both accessible and efficient for everyday users.

CCS Concepts: • **Computing methodologies** → **Artificial intelligence**; • **Information systems** → **Information retrieval**.

Keywords: Email Management, Artificial Intelligence, Natural Language Processing, Chrome Extension, Gmail API

ACM Reference Format:
Lin Yang, Ruoqian Huang, Longsheng Yan, and Ye Yu. 2025. Intelligent Email Assistant: An AI-Powered Email Management System. In . ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

In today's digital landscape, email remains a central mode of communication for both personal and professional use. However, with the volume of incoming messages growing steadily, users often struggle to keep their inboxes organized and to identify the most important information in a timely manner. This challenge is particularly evident in Gmail, one of the most widely used email platforms, where users receive a mix of work-related, promotional, and personal emails every day. To address this issue, we developed the *Intelligent Email Assistant*, a Chrome extension that integrates artificial intelligence into the Gmail user experience. The assistant automatically classifies emails by intent (e.g., urgent, business, spam), flags potential spam, and allows users to search their inbox using natural language queries. By leveraging OpenAI's large language models (LLMs) and the Gmail API, the system is capable of summarizing emails, suggesting user actions, and providing conversational access to email content. Unlike traditional rule-based filters, our assistant uses a hosted LLM to better understand context and intent in email content, enabling more flexible and accurate responses. This allows it to provide more accurate classifications and more helpful query responses, even when the user's request is vague or conversational in nature. The assistant is designed with usability in mind: it runs locally in the browser via a popup interface, communicates with a lightweight Flask backend, and operates in real time without requiring users to leave their Gmail tab.

1.1 Code Repository

The full source code and documentation for the project are available at: https://github.com/linyang6666/intelligent-email-assistant

2 System Architecture

The Intelligent Email Assistant adopts a modular client-server architecture that separates user interaction logic from backend processing and API communications. Figure 1 illustrates the high-level system design.

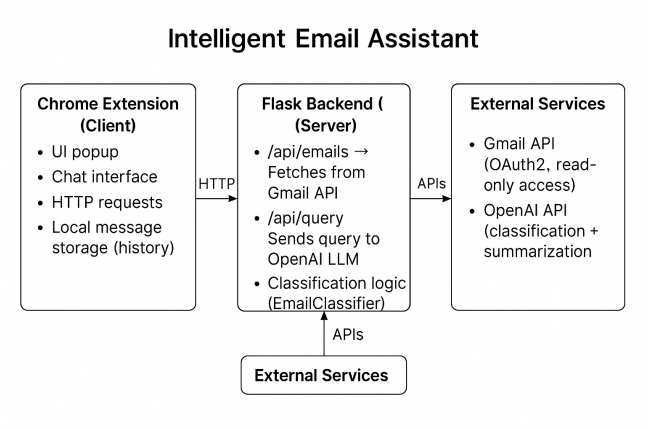

Figure 1. System Architecture of the Intelligent Email Assistant

2.1 Chrome Extension (Client)

The frontend of the system is implemented as a lightweight Chrome extension. It serves as the primary interface for users and is responsible for:

- Displaying a popup UI with an email preview list and a chatbot interface.
- Handling user input and sending natural language queries to the backend via HTTP requests.
- Rendering classification tags (e.g., Spam, Urgent) and suggested actions.
- Storing recent user messages locally to maintain conversational history across sessions.

2.2 Flask Server (Backend)

The backend is powered by a Flask web server that orchestrates all core logic and integrates external APIs. Its responsibilities include:

- Authenticating with the Gmail API via OAuth2 and fetching up to 100 recent emails.
- Caching email content and periodically refreshing data in the background.
- Running the `EmailClassifier` module to assign semantic tags such as `spam`, `business`, or `urgent`.
- Relaying natural language questions to OpenAI's LLM via the `AIProcessor`, and returning generated summaries or suggested actions.
- Returning structured JSON responses to the Chrome extension.

2.3 External Services

The system connects with the following external APIs:

- **Gmail API**: Provides read-only access to the user's inbox. It is used for fetching raw email metadata and message bodies.
- **OpenAI API**: Enables semantic classification and summarization of emails via GPT-based language models.

This architecture enables clear separation of concerns, allowing each component to be developed, tested, and improved independently.

3 Key Features and Implementation

3.1 Email Classification and Categorization

To help users triage emails efficiently, the system classifies incoming messages into high-level categories:

- **Urgent** — Time-sensitive or critical updates
- **Business** — Professional or work-related communication
- **Friendly** — Social, personal, or casual messages
- **Complaint** — Emails expressing dissatisfaction or issues

Classification is powered by OpenAI's GPT model, prompted with custom context that includes sender, subject, and body snippet for each email. The model returns structured tags in JSON format. If classification fails, the system defaults to a fallback label.

```
def classify_emails(self, emails, max_emails=20):
    emails_to_process = emails[:max_emails]
    context = self.build_prompt(emails_to_process)
    try:
        resp = self.client.chat.completions.create
    (
            model="gpt-4o-mini-2024-07-18",
            messages=[
                {"role": "system", "content": "You
     classify email intents."},
                {"role": "user", "content":
    context}
            ],
            response_format={"type": "json_object"
    }
        )
        result = resp.choices[0].message.content
        classifications = json.loads(result)["
    classifications"]
        return self._apply_classifications(
    emails_to_process, classifications)
    except Exception:
        return self._apply_default_classifications
    (emails_to_process)
```

3.2 Spam Detection

To reduce noise in the inbox, the system includes a lightweight spam detection module using a hybrid strategy:

- **Keyword-based Filtering** — Detects promotional and clickbait terms (e.g., "unsubscribe", "winner", "offer").

- **LLM-driven Classification** — Emails flagged as "spam" during the GPT-based classification step are also labeled accordingly.

```
def is_spam(self, email):
    spam_keywords = [
        "unsubscribe", "promotion", "deal", "limited time", "winner",
        "discount", "click here", "free", "offer"
    ]
    subject = email.get("subject", "").lower()
    body = email.get("body", "").lower()
    return any(kw in subject or kw in body for kw in spam_keywords)
```

This rule-based filter operates alongside GPT-based spam classification to ensure redundancy and improved accuracy.

3.3 Context-Aware Email Retrieval

Users can interact with their inbox via natural language queries like: "`Show me all security alerts`" or "`What looks like spam?`" The Flask backend parses the query and either matches keywords locally or forwards the context to OpenAI's LLM for semantic search.

```
@app.route('/api/query', methods=['POST'])
def process_query():
    query = request.json.get('query', '').strip()
    if 'spam' in query.lower():
        spam_emails = [e for e in classified_emails if e.get("is_spam")]
        return jsonify({'answer': format_spam_summary(spam_emails)})
    relevant = ai_processor.search_emails(email_cache, query)
    context = ai_processor.prepare_context(relevant, query)
    answer = ai_processor.query_openai(context, query)
    return jsonify({'answer': answer})
```

Responses include summaries and suggested actions formatted in a conversational tone.

3.4 To-Do Item Extraction

In addition to classification and query support, the system assists users by generating a concise to-do list extracted from recent email content. This feature prompts the LLM to summarize action items based on sender, subject, and snippet context. It helps surface deadlines or follow-up tasks without requiring users to read every message in detail.

```
def generate_todo_list(self, emails, max_items=5):
    """
    Generate a concise To-Do list based on the most recent emails.
    """
    # Build the prompt with up to 10 email summaries
    prompt = f"Here are the recent email summaries. Extract the top {max_items} action items as a numbered list (one per line):\n\n"
    for idx, email in enumerate(emails[:10], start=1):
        snippet = email['body'][:150].replace('\n', ' ')
        prompt += (
            f"{idx}. From: {email['sender']}, Subject: {email['subject']}, "
            f"Snippet: {snippet}...\n"
        )
    prompt += (
        "\nPlease return only the numbered To-Do list, for example:\n"
        "1. Reply to Alice about the project update\n"
        "2. Schedule a meeting with Bob regarding the budget\n"
    )

    return self.query_openai(prompt, "")
```

Tasks are presented in a collapsible checklist component within the UI, and users can mark completed items directly. This feature was particularly appreciated in user feedback for improving focus and inbox utility.

4 User Interface

The Gmail Assistant extension provides a streamlined and intuitive interface built for seamless integration within the browser. Key features are illustrated below using real usage examples.

- **Email Display with Tags:** Recent emails are shown in a scrollable panel, each marked with a category tag—such as `Urgent`, `Spam`, or `Business`—using colored badges and emoji icons for easy scanning.

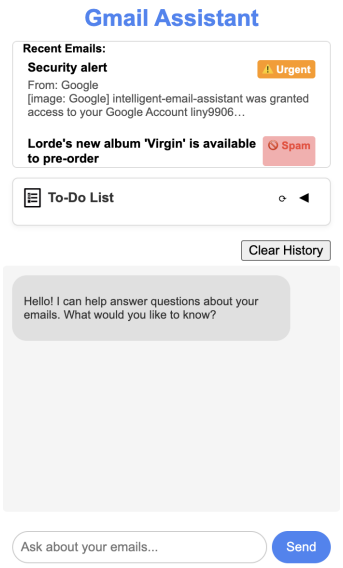

Figure 2. Email list showing classification tags (Urgent, Spam).

- **Natural Language Input Box:** Users can ask questions like "Show me all spam emails" or "Did I receive any alerts?" The assistant interprets the query and responds with relevant results.

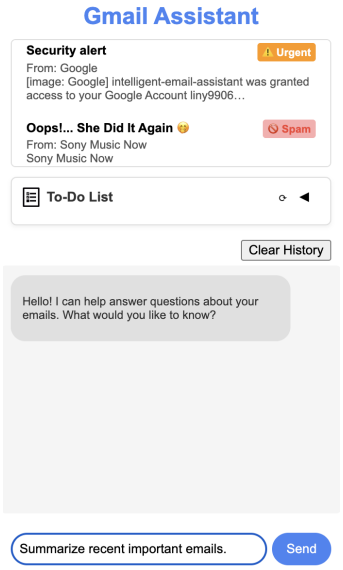

Figure 3. Spam-related natural language query and structured response.

- **Conversational Thread:** The assistant displays each query and response in a sequential chat interface, allowing users to track ongoing conversations and follow up naturally.

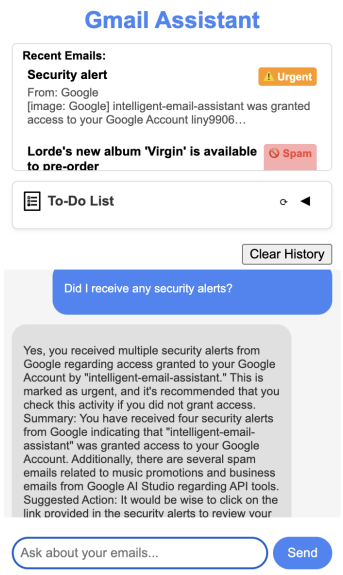

Figure 4. Chat-like interface with GPT-generated summary.

- **Structured Responses:** For queries such as spam filtering or security alerts, the assistant presents grouped results with sender names, subjects, and snippets arranged in a readable paragraph-style layout.

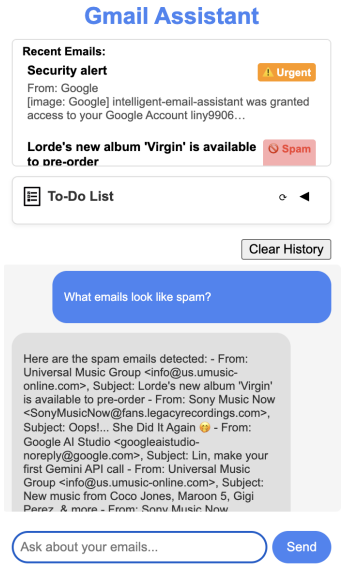

Figure 5. Assistant response showing detected spam email details.

- **Actionable Suggestions:** The assistant identifies emails with tasks or reminders and surfaces them as a checklist for follow-up.

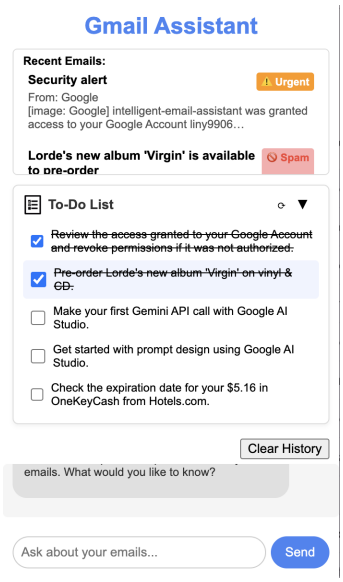

Figure 6. To-do list showing extracted actionable items from email.

- **Session Reset and Summaries:** A `Clear History` button resets the chat, while summary queries return a concise overview of recent messages and suggestions.

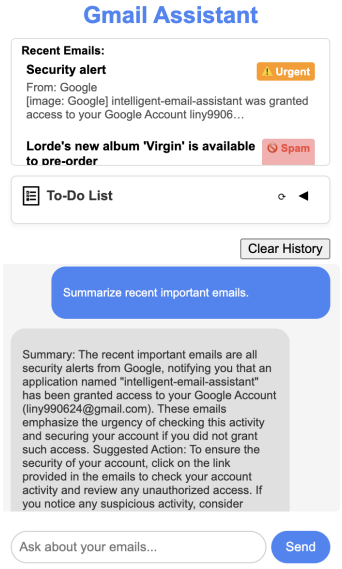

Figure 7. Summary of important emails with suggested actions.

This lightweight interface enables users to manage their inbox more intelligently—surfacing key insights and reducing the need to navigate away from Gmail.

5 Evaluation

To evaluate the performance and usability of the Intelligent Email Assistant, we conducted both internal testing and external user surveys. This hybrid approach allowed us to assess the system from both technical and user-centered perspectives. We acknowledge that our manually labeled dataset was limited in size and not statistically representative. However, it served as a practical resource throughout the development process—enabling rapid iteration, early debugging, and targeted refinement of key features.

5.1 Classification and Spam Detection Accuracy

As part of our developer-side validation, we manually labeled a sample of 20 emails from a real Gmail inbox and compared these labels against the system's predictions. While we acknowledge that 20 samples is a relatively small dataset, this self-test allowed us to probe failure cases and establish a baseline for accuracy during the development cycle.

- **Classification Accuracy**: 90.0%
- **Spam Detection Accuracy**: 80.0%
- **Average Response Time**: 3.33 seconds (batch of 20)
- **Processing Rate**: 360.5 emails/minute
- **Peak Memory Usage**: 70.0 MiB

Most classification errors involved borderline promotional content miscategorized as business or urgent. For example:

- "Your $5.16 in OneKeyCash is about to expire" was flagged as `urgent` instead of `spam`.
- "Changes to our One Key loyalty program" was misclassified as `business`.

Figure 8. Console output from manual evaluation showing prediction outcomes and system stats.

5.2 User Survey Feedback

We distributed a structured feedback form to 11 users, including both students and business professionals, to understand perceived usefulness, interface experience, and opportunities for improvement.

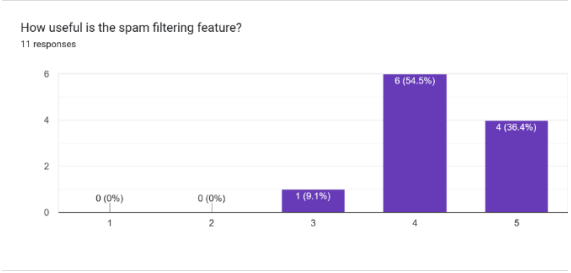

Figure 9. Participant demographics: 73% students, 27% business professionals.

Feature-specific scores were strong across the board:

- **Spam Filtering**: Avg score = 4.5/5
- **Email Categorization**: Avg score = 4.1/5
- **Chatbot Retrieval**: Avg score = 4.4/5
- **To-Do List**: Avg score = 4.6/5

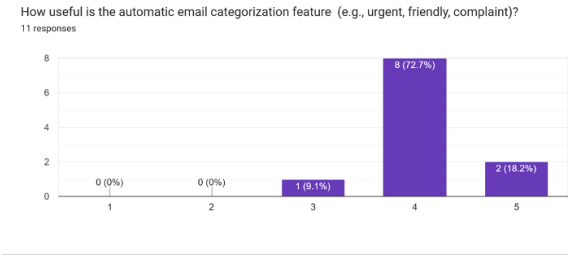

Figure 10. Perceived usefulness of email categorization.

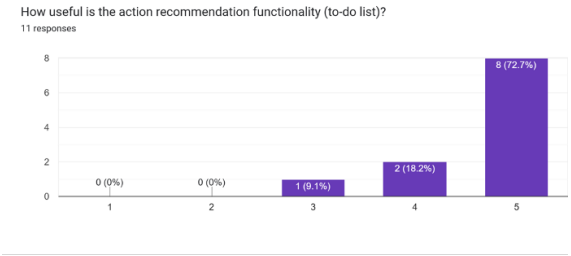

Figure 11. Usefulness ratings for the action recommendation feature.

Usability was also well received:

- **Interface Intuitiveness**: 100% rated 4 or 5
- **Responsiveness**: 100% rated 4 or 5
- **Time Saved**: 82% said Yes, 18% Maybe

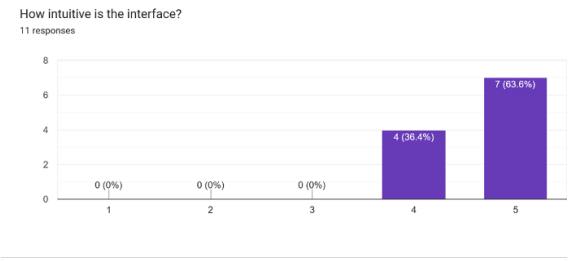

Figure 12. Intuitiveness of the assistant interface.

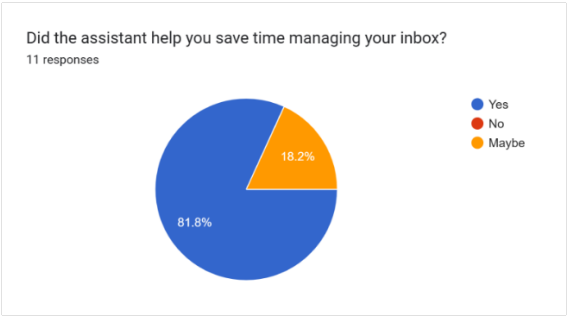

Figure 13. Majority of users report saving time with the assistant.

5.3 Qualitative Insights

Open-form responses added color to the quantitative ratings. Highlights include:

- Most-valued features: To-do list, chatbot summarization, classification clarity
- Suggested improvements: more robust formatting, better support for vague queries
- Feature requests: calendar integration, dark mode, support for Gmail threading

What feature did you find the most valuable, i	What didn't work as expected or needs impro	Any suggestions for new features or changes
To-Do list	reponse formatting can be improved (most of the	Add calendar integration for scheduling replies or
Todo List	None	None
I liked the automatic categorization — it made it re	Sometimes the spam filter misses promotional er	Would love dark mode and Gmail threading suppo
The to-do list from emails really helped me stay o	The chatbot sometimes misunderstood vague qu	Would be great to integrate with Google Calendar.
chatbot	More beautiful interface	None
The chatbot responses felt very natural.	The UI felt a bit cramped on smaller screens.	Add a "summarize thread" function.
The categorization tags are super helpful — I can	Maybe show more than 10 emails in the preview l	Add a search bar to manually override queries.
The assistant's ability to summarize emails in one	Classification sometimes misses borderline emai	Add integration with work calendars or Slack.
Todo List	None	Automatic notification function
Chatbot		Support more languages

Figure 14. Sample free-text survey feedback from participants.

5.4 Discussion

While our testing dataset was small and user pool modest, these evaluations reflect a development-phase system under

real usage. Feedback suggests the assistant delivers meaningful productivity gains and is well-aligned with user needs. We view these results as promising and intend to broaden evaluation efforts with larger datasets and more diverse users in future work.

6 Usage Guide

6.1 Installation

To install and run the Intelligent Email Assistant, follow these steps:

1. **Clone the repository:**

```
git clone https://github.com/linyang6666/intelligent-email-assistant.git
cd intelligent-email-assistant
```

2. **Set up the Python backend:**
   - Navigate to the `server/` directory.
   - Create and activate a virtual environment (recommended).
   - Install dependencies:

```
pip install -r requirements.txt
```

   - Export your OpenAI key as an environment variable:

```
export OPENAI_API_KEY=sk-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

   - Run the Flask backend:

```
python background.py
```

3. **Authenticate with Gmail:** On first run, the backend will print a Google OAuth URL. Open it in your browser, sign in, and grant access to the Gmail API. Once authorized, the assistant will cache and classify your most recent 100 emails.
4. **Load the Chrome extension:**
   - Open `chrome://extensions` in Google Chrome.
   - Enable `Developer mode`.
   - Click `Load unpacked` and select the `extension/` folder in the repo.

6.2 Basic Usage

Once the extension is installed and the server is running:

1. Click the Gmail Assistant icon in the Chrome toolbar.
2. The popup will display recent emails, each labeled with a color-coded tag (e.g., Urgent, Spam).
3. Type natural language questions such as:
   - `Did I receive any security alerts?`
   - `What emails look like spam?`
   - `Summarize recent important emails.`
4. The assistant will query your emails and respond in a chat-like format, often including summaries and suggested actions.
5. Click `Clear History` to reset the chat.

7 Future Work

While the Intelligent Email Assistant offers practical value in its current form, there are still several areas where we believe the system could be meaningfully improved.

- **Improved Spam Detection via Fine-Tuned Models**: Our current approach combines keyword heuristics with LLM-based predictions, which works reasonably well in most cases. However, it can mislabel edge cases such as promotional content or newsletters. Fine-tuning on a domain-specific labeled dataset could help reduce false positives and improve precision.
- **Richer Contextual Summarization**: Although the assistant can extract suggested actions and provide brief summaries, it does not yet capture broader conversational context—such as multi-message threads or implicit follow-ups. Incorporating more context-aware summarization would allow for deeper understanding and more helpful user prompts.
- **Broader Provider Integration**: Currently, the system is designed specifically for Gmail. This decision helped streamline development but limits its adoption. In future iterations, we hope to support other email providers such as Outlook or Yahoo through IMAP or official APIs.
- **Scalable Evaluation Framework**: Much of our evaluation relied on a small set of manually labeled emails and user testing among our developers and volunteers. While this allowed for iterative improvements, a larger and more diverse dataset—either synthetic or anonymized—would provide a stronger foundation for validating accuracy and performance.
- **Dark Mode Support**: One user mentioned the lack of a dark mode as a usability concern. Adding support for dark-themed UI would improve comfort, especially for users working at night or in low-light environments.

8 Conclusion

This work presents the Intelligent Email Assistant, a Chrome-based email management tool that applies state-of-the-art language models to everyday productivity. By combining LLMs, Gmail APIs, and an intuitive frontend, the assistant enables users to classify, filter, and query their inbox using natural language. Our evaluation shows promising classification and spam detection performance with reasonable latency and memory usage. However, limitations remain in generalization and dataset size. The absence of large-scale labeled emails constrains both model validation and performance tuning, and prompts the need for better infrastructure

and data availability in future iterations. Despite these constraints, the system demonstrates the practical potential of conversational AI in email workflows. We believe this work contributes a valuable foundation for building smarter, more assistive email clients, and opens doors for future research in intent understanding, summarization, and personalized email interaction.

References

[1] OpenAI. (2023). gpt-4o-mini-2024-07-18. https://openai.com
[2] Google. (2023). Gmail API. https://developers.google.com/gmail
[3] Google. (2023). Chrome Extensions. https://developer.chrome.com/docs/extensions