

COMP4801

Final Year Project

Universal GenAI Agent – Integrated Interactive Intelligence

Final Report

Group: FYP24100



The University of Hong Kong

Department of Computer Science

Supervised By:

Prof. Tao Yu

Student:

Fung Ho Sang (3035927386)

Date of Submission: 21st April 2025

Abstract

This project addresses the pervasive issue of information overload, a common challenge in today's digital world where individuals and organizations struggle to obtain and process large amounts of data every day. The Universal GenAI Agent aims to alleviate this problem by developing an intelligent content delivery system that automates content retrieval from selected sources and uses Google's Gemini GenAI for summarization. By providing personalized, concise updates in real time, the system reduces the time and effort required to process information while enhancing user efficiency and decision-making. The Universal GenAI Agent empowers users to stay informed by providing timely, summarized updates based on their specific preferences and interests. This is particularly beneficial in real-world scenarios such as job searching, where users can receive notifications about job openings and company updates, or in industry news monitoring scenarios, where timely and processed insights can drive informed decisions. Preliminary results suggest that the Universal GenAI Agent can integrate multiple services. By automating content extraction and summarization, the system significantly reduces manual processes required in information retrieval, saves users valuable time and delivers actionable insights. Its scalability and adaptability make it suitable for a wide range of applications beyond job seeking, including tracking travel alerts, monitoring industry updates, or staying updated on personal interests. By addressing the inefficiencies of traditional information retrieval processes, this project offers a practical and innovative solution to information overload. It not only enhances the user experience but also highlights the potential of AI-driven personalization in delivering smarter, more efficient content solutions.

Table of Contents

Acknowledgement	i
List of Figures	ii
List of Tables	iv
List of Abbreviations	v
1. Introduction.....	1
2. Methodology	3
2.1 System Architecture Design	3
2.2 Engineering choices, decisions and justifications.....	5
2.2.1 Discord Chat Interface	5
2.2.2 Cron Job Notification System.....	5
2.2.3 Master Server	6
2.2.4 Web Scraping and Content Extraction.....	6
2.2.5 Third-party API connectors	6
2.2.6 NeDB Database.....	7
2.2.7 GenAI Integration for Summarization	7
2.3 Challenges and Mitigations.....	8
2.3.1 Web Scraping Challenges	8
2.3.2 System Reliability	9
3. Real-world Usages	10
3.1 Retrieving Job Postings.....	10
3.2 Personal Interest Tracking	13
3.3 Web Source Content Retrieval.....	15
3.4 Clear Stored Preferences	18
4. Final Progress and Future Enhancement Ideas	20
5. Conclusion	22
References	23

Acknowledgement

I would like to express my gratitude to Prof. Tao Yu for his supervision of this project, as well as the valuable comments and feedback on the project design and implementation.

List of Figures

Figure 1: Workflow sequence illustrated by arrows and numbered steps. The process either begins with user input in the Discord Chat Interface (1a) or an automated cron job that regularly notifies the user (1b), then the master server will handle the request and proceed through the following steps: (2) Extract raw data from either website content extracted using Selenium or third-party data from API service providers, (3) Retrieve user detail from NeDB database, (4) Analyze and personalize content using Gemini GenAI, and (5) Output digested content to the user in the Discord Chat Interface.....	4
Figure 2: Guide to execute the /jobs command, by typing /jobs in the chat to retrieve relevant job postings.	10
Figure 3: Example response after the command /jobs has been executed, showing job titles, company names, and other related details. The blue “Apply Here” hyperlink redirects to the LinkedIn page to apply for the job. In this example, no user preferences have been set, so the result may not be relevant.	10
Figure 4: Guide to execute the /set_jobs command, by typing /set_jobs in the chat to set job preferences.	11
Figure 5: Example question and answering interaction of the /set_jobs command that sets job preferences and cron job frequency. This interactive approach is designed to make the process easy to follow.	11
Figure 6: Example response after the command /jobs has been executed.	12
Figure 7: Guide to execute the /cat command, by typing /cat in the chat to retrieve a random cat image.	13
Figure 8: Example response after the command /cat has been executed, the API gives a random cat image.	13
Figure 9: Guide to execute the /set_cat command, by typing /set_cat in the chat to configure cat command.	14
Figure 10: Example question and answering interaction of the /set_cat command that sets cat preferences of ragdoll breed and cron job frequency of delivering a cat picture every minute.	14
Figure 11: Example output from the cron job every minute from 11:20am to 11:25am according to user preference set in Figure 9 (ragdoll breed, deliver every minute).	15

Figure 12: Guide to execute the /analyze command, by typing /analyze [url] and using “scmp.com” in the url field as an example website to scrape content	16
Figure 13: Example response after the command /analyze has been executed with url “scmp.com”	16
Figure 14: Example response after asking a follow-up question to the previous response of /analyze	16
Figure 15: Guide to execute the /set_analyze command, input /set_analyze to set analysis preferences.	17
Figure 16: Example question and answering interaction of the /set_analyze command that sets user’s custom web source to retrieve content for automated digested delivery (e.g. scmp.com), the user’s topics of interests (e.g. geo-political, history, geography), and the cron job frequency (e.g. every minute).	17
Figure 17: Example output from the cron job of /analyze according to user preference set in Figure 16. The content has been personalized to the user’s set preferences, with most of the summarized content and suggested questions to ask being related to the geo-political topic that matches the user’s interest.	18
Figure 18: Guide to execute the /clear_preferences command, by typing /clear_preferences in the chat.	19
Figure 19: The confirmation sent to user after successful removal of the user’s data.....	19

List of Tables

Table 1: Comparison between OpenAI’s ChatGPT API and Google’s Gemini API	7
--	---

List of Abbreviations

Abbreviation	Full Form
API	Application Programming Interface
BSON	Binary JavaScript Object Notation
CAPTCHAs	Completely Automated Public Turing Test to Tell Computers and Humans Apart
GenAI	Generative Artificial Intelligence
LLM	Large Language Model
JSON	JavaScript Object Notation
RSS	RDF Site Summary

1. Introduction

Information overload is a pervasive issue in today's digital age, affecting both individuals and organizations. Information overload refers to the overwhelming amount of data that negatively affects decision-making, productivity, and mental well-being [1]. Research by OpenText highlights the scale of this issue: 80% of respondents report experiencing information overload, with 27% needing to access more than 11 tools daily—up from 15% two years ago. Nearly half spend over an hour daily searching for data, while 42% find critical information scattered across sources. [2]. An evident example of this challenge is the job-seeking process. Job seekers often maintain a targeted list of companies but face significant difficulty monitoring career pages and tracking relevant news across multiple platforms. This fragmented flow of information not only leads to decision fatigue and reduces the efficiency of their job search efforts, but also very time consuming. Moreover, the lack of standardized tools, such as RSS (RDF Site Summary) feeds, hinders automated information retrieval. Users must manually monitor updates, increasing the risk of missing critical information.

With the recent development in GenAI (Generative Artificial Intelligence) tools that utilize Large Language Models (LLM) to understand texts and output responses, and the wide range of Application Programming Interface (API) available as specialized data sources, they may be combined to form an analyzed content delivery system. This Universal GenAI Agent project aims to address the challenges of information overload by developing a personalized content delivery system that automates content retrieval and uses GenAI to summarize content into a concise, digested form. The motivation for this project stems from the need to alleviate the cognitive burden of searching through vast amounts of information. By integrating advanced content retrieval, analysis, and personalization, the system can transform the vast amounts of information into highly relevant, context-aware, and user-specific outputs.

There are three main objectives to this project:

- 1) To develop an all-Purpose, GenAI-embedded agent that integrates seamlessly with a wide range of third-party services, delivers tailored content to users regularly and allows easy user interaction to efficiently perform routine actions.
- 2) To enhance user engagement and satisfaction by providing personalized information.
- 3) To explore new use cases of GenAI in content delivery that can improve efficiency.

The main deliverable for this project is the development of the Universal GenAI Agent, a comprehensive software system designed to address information overload through intelligent content delivery. The system integrates six key components: a web scraping module capable of real-time data extraction, a third-party API connector to obtain usage-specific data, a GenAI service connector for personalized content summarization, a database for storing user-specific preferences, a notification system to deliver updates for multiple users, and a Discord AI Chatbot agent for easy interactions. Completed course deliverables include a detailed project plan, a project web page with regular progress updates, an interim report, this final report, and the first and final presentations.

Similar research shows significant improvements in user engagement through AI-driven personalization. Study by Sodiya et al. [3] illustrates that AI-driven personalization can boost user engagement by presenting relevant information based on user behavior, significantly improving overall satisfaction. Research by Smith et al. [4] indicates that AI systems can analyze user behaviors to provide personalized recommendations, saving time and minimizing the effort needed to retrieve relevant content. These studies suggest potential advantages of implementing a personalized content delivery system. Despite these promising findings, there remains a gap in the practical application of AI-driven personalization. The current studies primarily focused on e-commerce and entertainment rather than real-time information retrieval, and very few studies specifically explore dynamic contexts such as job hunting or industry news monitoring, where users need constantly updated, summarized content. Existing systems also tend to be static, lacking the flexibility to adapt to users' evolving needs. The Universal GenAI Agent addresses this gap by integrating real-time web scraping and GenAI-powered summarization to deliver concise, personalized content in real time, reducing information overload and enhancing user efficiency.

In the remaining sections, Section 2 outlines the project methodology, with details of the technologies implemented and design decisions made during the development of the Universal GenAI Agent. Section 3 highlights the real-world use cases, showcasing the system's potential in real-time content retrieval and summarization. Section 4 explores new feature ideas for future enhancement. Finally, Section 5 provides the conclusion, summarizing the key contributions of the project and its potential implications for addressing information overload through AI-driven solutions.

2. Methodology

In Section 2, there are three subsections: Section 2.1 gives the overall design of system architecture and illustrates the logic flow between different components in the Universal GenAI Agent. Section 2.2 justifies the specific design choices and decisions made to each component. Section 2.3 illustrates the actions that have been taken to mitigate potential challenges during system operation.

2.1 System Architecture Design

The Universal GenAI Agent adopts a modular architecture, designed to support efficient development and precise debugging. The system comprises seven distinct components, each performing a specific function, listed sequentially according to their roles within the overall workflow below:

- 1) Discord Chat Interface: Serves as the user interface, allowing users to request information, interact with the system, and receive responses in real time.
- 2) Cron Job Notification System: Regularly executes the workflow to push summarized content to users at specified intervals, ensuring timely updates through the Discord Chat Interface.
- 3) Master Server: The input/output point that connects the discord interface, and acts as the central coordinator managing communication and operations between various components to ensure smooth workflow execution.
- 4) Selenium Web Scraping Module: Automates the retrieval of content from selected websites, including dynamic content rendered through JavaScript, to ensure comprehensive data collection.
- 5) Third-party API Connectors: Acts as another possible data source other than websites. These connectors retrieve highly relevant data from third-party sources for different specialized use cases, such as the tracking of a specific interest.
- 6) NeDB Database: Stores user preferences, such as topics of interest and notification settings, as well as previously analyzed content, enabling personalized content delivery.
- 7) Gemini GenAI Service: Processes the retrieved content and generates concise, user-relevant summaries based on user preferences.

Figure 1 below shows the workflow between different components when the system handles a user input.

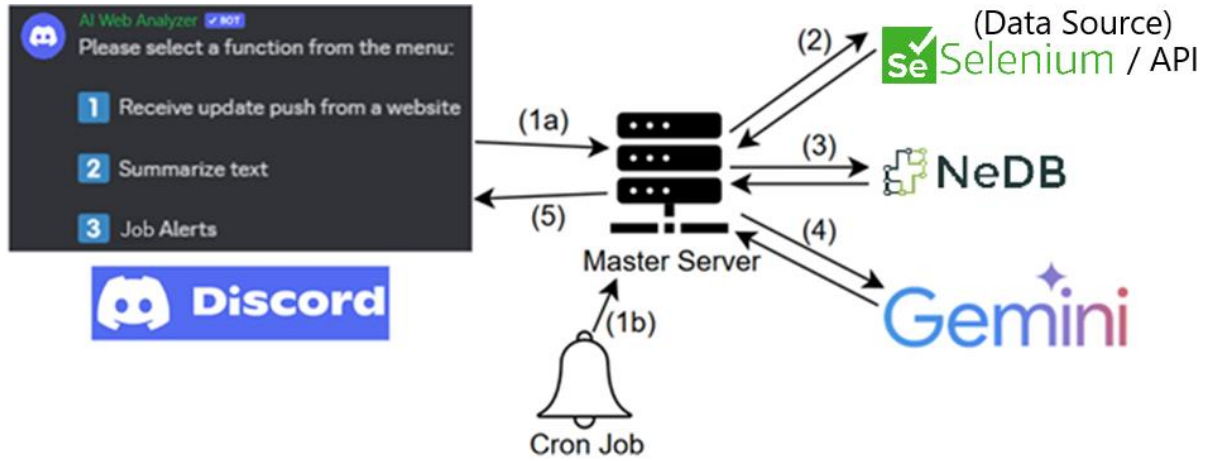


Figure 1: Workflow sequence illustrated by arrows and numbered steps. The process either begins with user input in the Discord Chat Interface (1a) or an automated cron job that regularly notifies the user (1b), then the master server will handle the request and proceed through the following steps: (2) Extract raw data from either website content extracted using Selenium or third-party data from API service providers, (3) Retrieve user detail from NeDB database, (4) Analyze and personalize content using Gemini GenAI, and (5) Output digested content to the user in the Discord Chat Interface.

The workflow of the Universal GenAI Agent (Figure 1) begins with either user input through the Discord Chat Interface (Step 1a) or an automated process initiated by the cron job notification system (Step 1b), which delivers updates based on predefined schedules or user preferences. The Master Server handles the requests and manages communication between the components. The process proceeds by obtaining the raw content to be analyzed, from either user selected websites scraped using Selenium in real time, or third-party data from API service providers (Step 2). These raw contents are then combined with the user's preferences retrieved from the NeDB database, including topics of interest, language settings, and recent queries (Step 3) to personalize the contents with individual requirements in subsequent operations. The retrieved content, alongside user preferences, is processed by the Gemini GenAI service (Step 4), which analyzes the data, generates concise and relevant summaries, and personalizes the output to align with the user's specific needs. The final summarized content is delivered to the user through the Discord Bot (Step 5), notifying the users with the newest information. Users may further engage with the system via the Discord Chat interface to ask follow-up questions (the workflow starts again with Step 1a), to retrieve additional details. This workflow is designed to provide an efficient and highly personalized content delivery system while addressing the challenges of information overload.

2.2 Engineering choices, decisions and justifications

This section illustrates and explains the choice of the six core components that the Universal GenAI Agent is built on. These components were chosen based on their technical capabilities, ease of development, and alignment with the overall goals of the system. Each subsection below would discuss the choice of each component.

2.2.1 Discord Chat Interface

The Discord Chat Interface serves as the primary user interface for the Universal GenAI Agent, enabling users to interact with the system in real time. Discord was chosen due to its popularity as an instant messaging software, providing web, mobile app and PC versions that allow users to receive notification in their desired devices. Compared to traditional web-based interfaces, Discord provides a familiar platform for users, reducing onboarding time and simplifying communication in the format of daily messaging. This interface allows users to submit queries, receive personalized content, and interact with the AI agent for follow-up questions, creating a dynamic and engaging user experience. Its real-time messaging capabilities also ensure that users can access updates and responses quickly and efficiently. Discord provided a robust API with clear documentation of the data models [5], allowing the Discord AI agent to interact with the master server.

2.2.2 Cron Job Notification System

The Cron Job Notification System has been implemented to deliver summarized content to users at specified intervals. Cron jobs enable automated, time-based task scheduling. The system will execute the workflow at predefined intervals (e.g. every minute, every hour, daily) to continuously monitor data sources for content updates. When new or relevant content is detected by the web scraping module, notifications will be triggered in real time, ensuring that users are well notified without requiring the mental strain of actively searching for information. This system is implemented using Node-Cron, a comprehensive Node.js library that facilitates the scheduling of tasks at specified intervals. By integrating seamlessly with the Node.js architecture of the Universal GenAI Agent, Node-Cron leverages the non-blocking nature of Node.js, ensuring that scheduled tasks do not disrupt the main application flow. In practice, this means that each cron job can have its own interval defined based on user preferences and the requirements of data sources, and execution of the cron jobs will not disrupt the execution needs of other modules, such as the real-time input/output needs for the Master Server below.

2.2.3 Master Server

The Master Server acts as the central coordinator. It listens to all user requests on Discord and coordinates the execution of these requests by managing communication and operations among different components of the system. This centralized architecture was selected to ensure smooth workflow execution and to simplify the integration of different modules. The Master Server directs data flow, initiating processes such as triggering web scraping or content summarization, and maintaining system reliability via detailed logging and robust error case handling. By centralizing control, the server maintains control of each sub-service and ensures that all components work cohesively. Additionally, the modular nature of the server design facilitates scalability, allowing for future expansion or integration of new features without disrupting the existing workflow.

The Master Server has been implemented using Node.js v.20.15.0, a JavaScript runtime widely used in the transfer of JSON (JavaScript Object Notation), the payload format used in communication between different components, to allow smooth data flow. Node.js was also chosen for its asynchronous, event-driven architecture, that efficiently handles multiple concurrent tasks [6]. The system requires parallel processing to process requests from multiple concurrent users and interaction with other components, such as web scraping, API calls, and database. The native support to asynchronous programming and non-blocking execution of Node.js is crucial for the Master Server to handle multiple tasks simultaneously.

2.2.4 Web Scraping and Content Extraction

Web scraping is a vital component of the Universal GenAI Agent, facilitating the automated extraction of relevant content from various websites. The web scraping algorithm is based on Selenium, as it can fetch dynamic content generated by JavaScript, simulating user interactions to access information that is not available in static HTML. This dynamic scraping process can effectively gather a wide range of content types, including job postings and articles, fulfilling existing requirements and allowing for future expansion of features that require complex content retrieval.

2.2.5 Third-party API connectors

Third-party API connectors provide access to specialized data sources that enhance the system's overall functionality, allowing for a broader range of use cases and enriching the content delivered to users. The decision to incorporate third-party APIs was motivated by the

need for reliable and diverse data sources beyond what web scraping alone can provide. These APIs can offer data that is more structured and often comes with built-in functionalities, such as filtering and searching capabilities. Examples of third-party APIs implemented include those for job listings, news, and specific interest tracking.

2.2.6 NeDB Database

The database is designed to store user-specific information essential for personalizing the content output for every user. This includes user IDs, user preferences such as topics of interest, keywords to look for, filtering preferences, notification frequency, and recent queries. By capturing these details, the system can deliver tailored content that aligns with the user's interests and preferred communication methods. The database structure is also designed to accommodate future expansion of additional usage-specific data, with each use case (e.g. job hunting or tracking of a specific personal interest) having its own set of preferences, ensuring flexibility as personalization needs evolve.

The database has been implemented using NeDB due to its lightweight, schema-less structure and compatibility with JSON data format that enables seamless data mapping with the JSON payload of API communication between system components, simplifying data storage and retrieval.

2.2.7 GenAI Integration for Summarization

The integration of GenAI utilizes Google's Gemini API, specifically the Gemini 1.5 Flash API [8]. The choice is a cloud-based GenAI instead of a local LLM. Locally run models still require significant computational resources such as high-end GPUs, especially for large models. Additionally, maintaining and updating the model, managing dependencies, and ensuring scalability can be challenging. Cloud-based models offer regular updates and eliminate hardware performance limitation, making them more practical for most use cases compared to running local models.

Cloud-based models may have different usage limits. Table 1 below shows the comparison in the rate limit and usage limit of the free tier between the API of OpenAI's ChatGPT-3.5-Turbo [7] and Google's Gemini 1.5 Flash [8].

Table 1: Comparison between OpenAI’s ChatGPT API and Google’s Gemini API [7-8]

	ChatGPT-3.5-Turbo API	Gemini 1.5 Flash API
Rate Limit	3 Requests per minute 200 Requests per day	15 Requests per minute 1,500 Requests per day
Usage Limit	40,000 Tokens per minute	1,000,000 Tokens per minute
Pricing	Free of charge	Free of charge

Referring to Table 1, Gemini has a much higher usage limit than that of ChatGPT, offering 1,000,000 tokens per minute, which is 25 times greater than ChatGPT’s 40,000 tokens per minute. This is the major reason for choosing Gemini. A larger number of tokens indicates longer text can be input to the model. As websites usually have very long texts, it is crucial that the GenAI model supports a long length of content. In this case, Gemini is vastly superior to ChatGPT. Additionally, Gemini supports a higher rate limit of 15 requests per minute and 1,500 requests per day, compared to ChatGPT’s 3 requests per minute and 200 requests per day. This higher rate limit ensures the system can handle multiple simultaneous user requests efficiently, reducing the risk of delays or bottlenecks. Combined with its superior token capacity, Gemini enables the Universal GenAI Agent to deliver consistent and reliable performance, even under heavy usage.

The Gemini connector has been developed using Node.js since communication with Gemini’s API server uses the JSON data structure. This choice also ensures consistency with other system components, simplifying integration and maintaining a unified development framework.

2.3 Challenges and Mitigations

While the project has mostly progressed smoothly, certain challenges have been identified during the development. These potential difficulties, along with their mitigation strategies, are outlined below.

2.3.1 Web Scraping Challenges

Dynamic websites often implement anti-scraping measures such as CAPTCHAs (Completely Automated Public Turing Test to Tell Computers and Humans Apart), rate limits, or JavaScript-based content rendering, which may hinder the effectiveness of the scraping module.

To address this, the web scraping module utilizes Selenium to simulate user interactions and retrieve dynamic content. Additionally, change of proxy and request delays have been implemented in case of a series of bad requests caused by website-imposed limitations. The system will also adhere to website terms of service and only scrape publicly available content to ensure ethical compliance.

2.3.2 System Reliability

There may be unexpected errors in modules such as web scraping, API communication, or the cron job system that could disrupt the overall workflow. To mitigate this, the Master Server will include robust error-handling mechanisms and detailed logging by the extensive use of ‘try-catch’ blocks to identify the point of failure and resolve issues promptly. Retry mechanisms have been implemented to ensure that temporary failures such as a missed API response or a network issue can be resolved, and the overall workflow can continue. The retry mechanisms will automatically attempt to reprocess failed tasks after a short delay, minimizing manual intervention for temporary issues. Regular monitoring of detailed logs with status of operation will help identify potential issues during operations, ensuring the system remains reliable.

3. Real-world Usages

This section introduces the Universal GenAI Agent project, showcasing its capabilities in optimizing information retrieval. There are three subsections: Section 3.1 discusses the `/jobs` command that fetches job listings from LinkedIn. Section 3.2 covers personal interest tracking by illustrating the `/cat` command that delivers regular cat pictures for users. Section 3.3 explains the `/analyze` command, enabling content analysis from any website with follow-up question capabilities. These example usages highlight the efficiency and user-friendly nature of these functionalities.

3.1 Retrieving Job Postings

- Command `/jobs`: Retrieves new job postings from LinkedIn API

The following figures illustrate the use of the `/jobs` function. Figure 2 demonstrates how to execute the command by simply typing `/jobs` to retrieve relevant job postings from the LinkedIn API. Figure 3 displays the response of `/jobs` command, showing the available job opportunities.

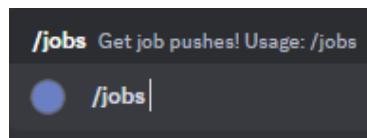


Figure 2: Guide to execute the `/jobs` command, by typing `/jobs` in the chat to retrieve relevant job postings.

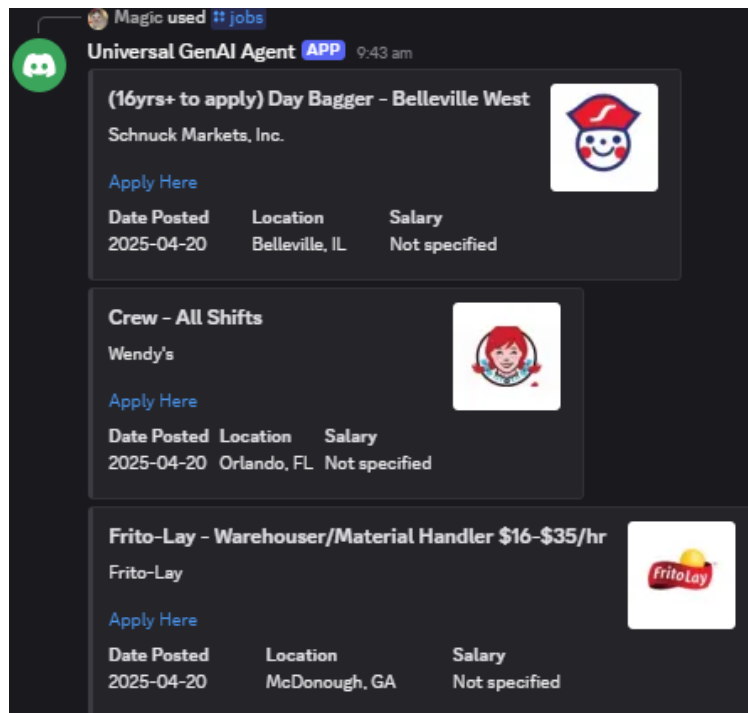


Figure 3: Example response after the command `/jobs` has been executed, showing job titles, company names, and other related details. The blue “Apply Here” hyperlink redirects to the LinkedIn page to apply for the job. In this example, no user preferences have been set, so the result may not be relevant.

- Command `/set_jobs`: Set job posting preferences

The following figures illustrates the use of the `/set_jobs` function. Figure 4 demonstrates how to execute the command. Figure 5 demonstrates the interactive steps to set user preferences and the cron job frequency.

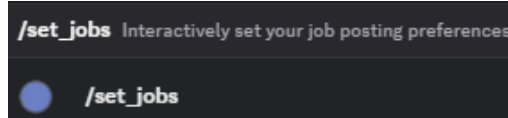


Figure 4: Guide to execute the `/set_jobs` command, by typing `/set_jobs` in the chat to set job preferences.

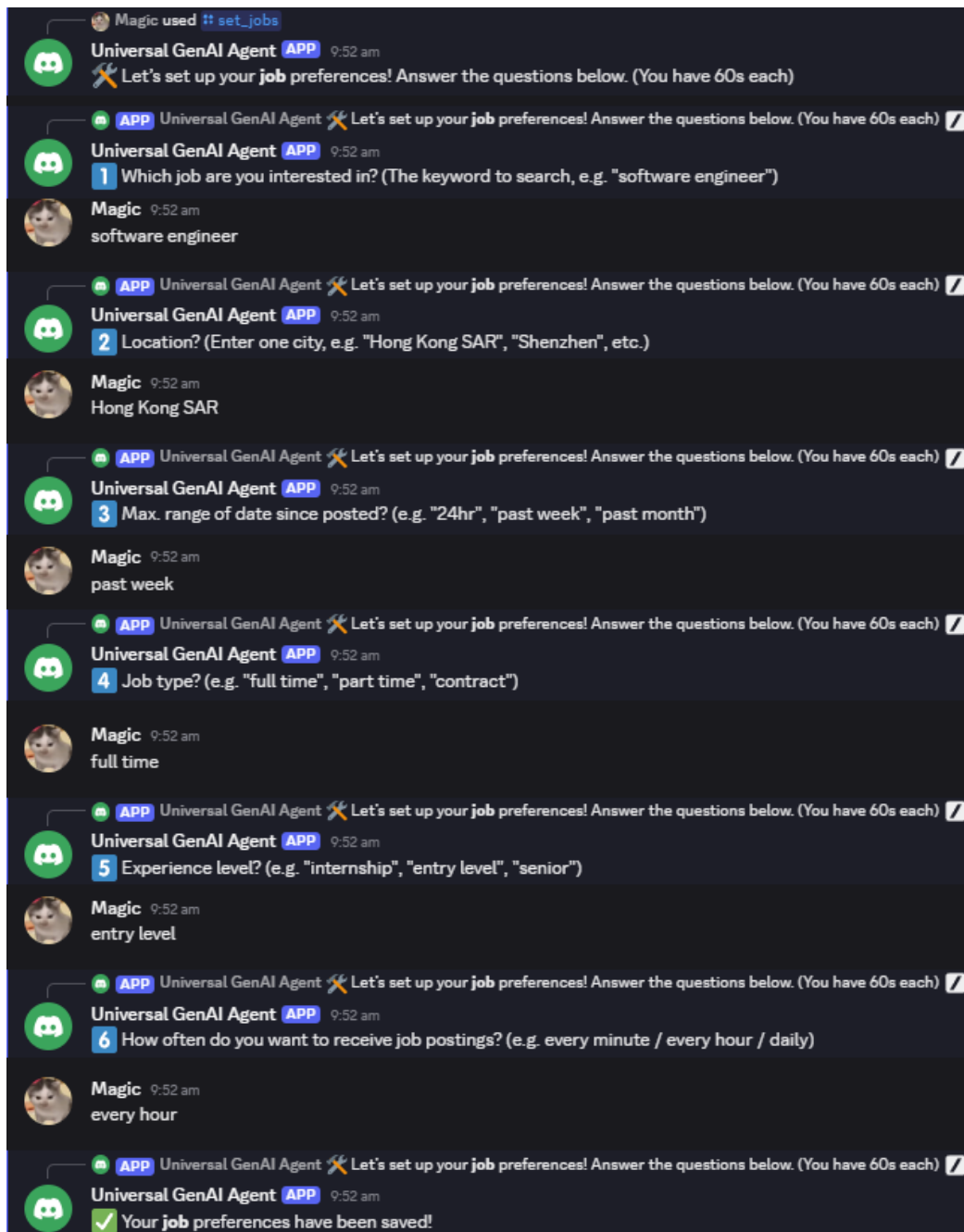
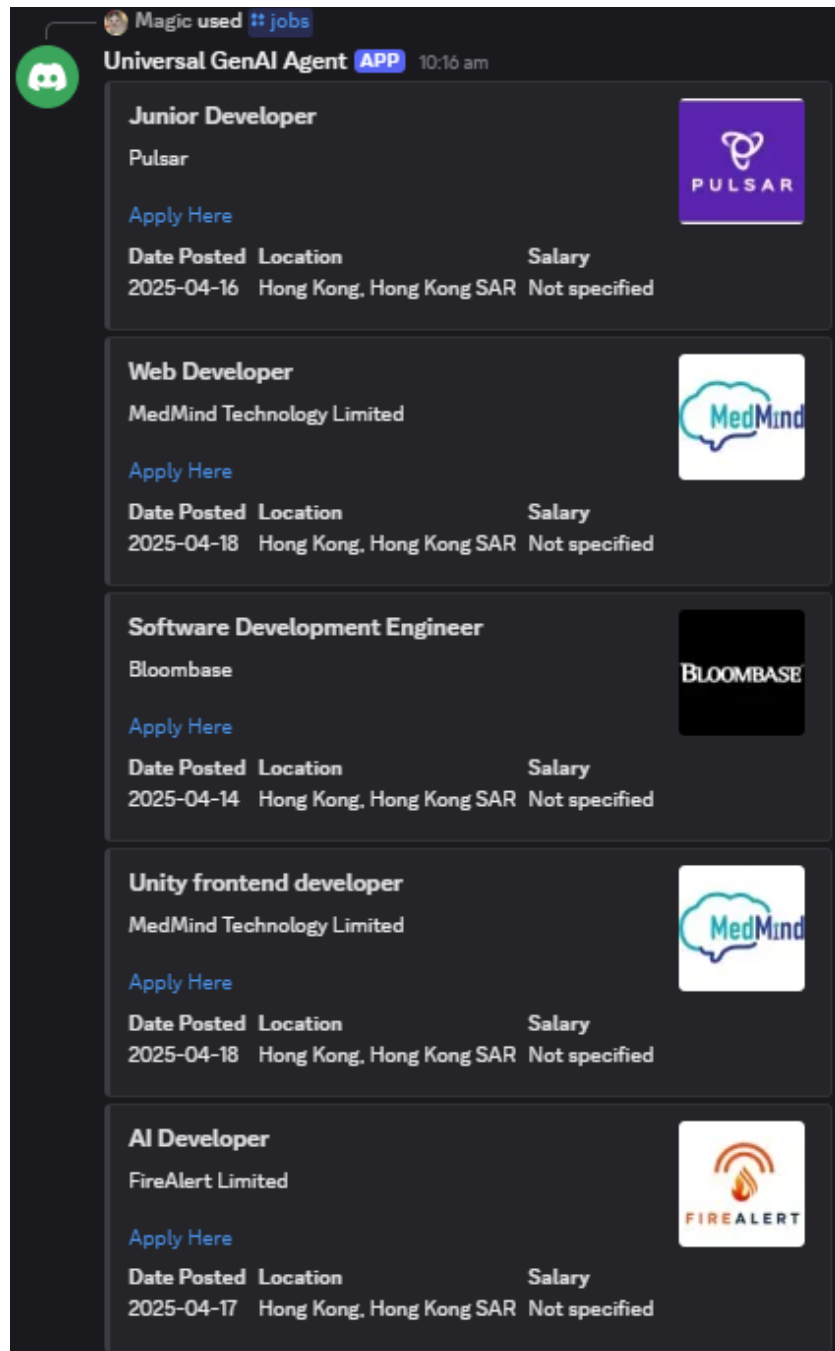


Figure 5: Example question and answering interaction of the `/set_jobs` command that sets job preferences and cron job frequency. This interactive approach is designed to make the process easy to follow.

- Re-run command /jobs after preferences have been set

Figure 6 displays the response received after executing /jobs command, with the user preferences set in Figure 5 above using the /set_jobs command.



The screenshot shows a Discord chat interface. At the top, the chat header includes a profile picture of a green circle with a white robot icon, the name 'Universal GenAI Agent' with a blue 'APP' tag, and the time '10:16 am'. Above the chat area, it says 'Magic used :: jobs'. The chat area contains five job listings, each in a dark-themed card. Each card has a job title, a company name, a company logo, a blue 'Apply Here' link, and a table with three columns: 'Date Posted', 'Location', and 'Salary'.

Job Title	Company	Date Posted	Location	Salary
Junior Developer	Pulsar	2025-04-16	Hong Kong, Hong Kong SAR	Not specified
Web Developer	MedMind Technology Limited	2025-04-18	Hong Kong, Hong Kong SAR	Not specified
Software Development Engineer	Bloombase	2025-04-14	Hong Kong, Hong Kong SAR	Not specified
Unity frontend developer	MedMind Technology Limited	2025-04-18	Hong Kong, Hong Kong SAR	Not specified
AI Developer	FireAlert Limited	2025-04-17	Hong Kong, Hong Kong SAR	Not specified

Figure 6: Example response after the command /jobs has been executed.

The response by executing /jobs command is identical to the cron job response as they share the same logic. In the response, the user preference filtering has been applied, returning jobs related to the key word “software engineer”, the posted dates within the specified “past week”, and the job location being “Hong Kong SAR”. By providing these customization options, results can closely align with user requirements and users can obtain information they need. In

the case of job searching, the system allows timely access to the latest job openings and easy-to-read descriptions, effectively reducing information overload and cognitive effort.

3.2 Personal Interest Tracking

This subsection explores how the Universal GenAI Agent enhances user engagement through personalized interest tracking and makes the experience more enjoyable and relevant. There are The primary command to illustrate this feature is `/cat`, which retrieves a random cat picture from The Cat API. This command serves as a fun and relaxing way for users to enjoy content that reflects their love for cats. It illustrates the agent's capability to cater to personal interests, providing a quick and delightful visual experience.

- Command `/cat`: Retrieves a random cat image from third-party “The Cat API”

The following figures illustrate the use of the `/cat` function. Figure 7 demonstrates how to execute the command by simply typing `/cat` to retrieve cat images. Figure 8 displays the response of the `/cat` command.

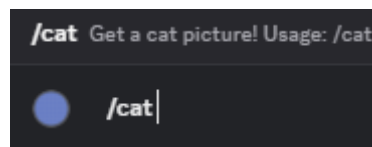


Figure 7: Guide to execute the `/cat` command, by typing `/cat` in the chat to retrieve a random cat image.

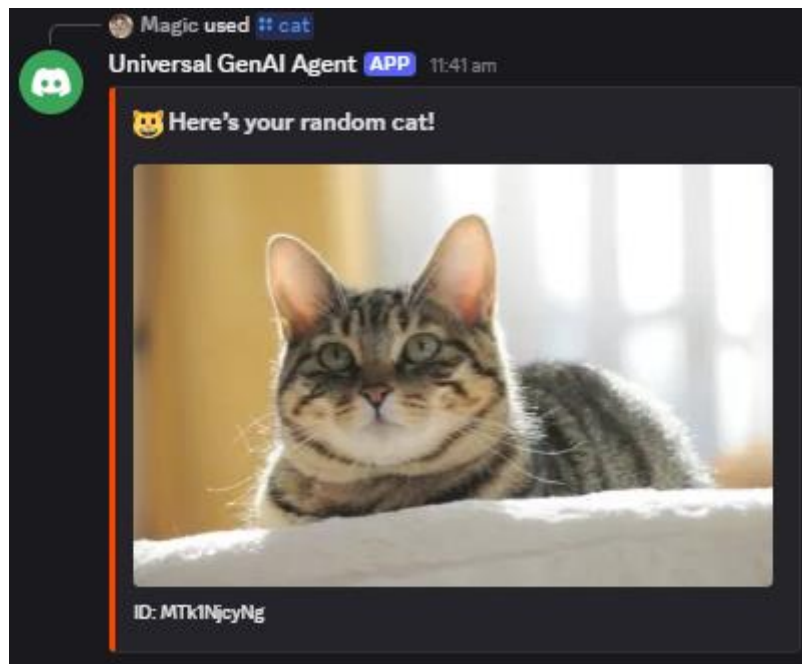


Figure 8: Example response after the command `/cat` has been executed, the API gives a random cat image.

To achieve a higher level of customization to cater to different users' preferences, command `/set_cat` can allow scheduled delivery of cat images and even select the breed of the cat to retrieve a user's most preferred cat image.

- Command `/set_cat`: Set the preferred breed and delivery frequency of cron job.

Figure 9 below demonstrates how to execute the command `/set_cat`. Figure 10 below demonstrates the interactive steps to set user's cat breed preferences and the cron job frequency.

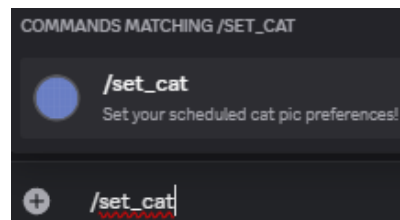


Figure 9: Guide to execute the `/set_cat` command, by typing `/set_cat` in the chat to configure cat command.

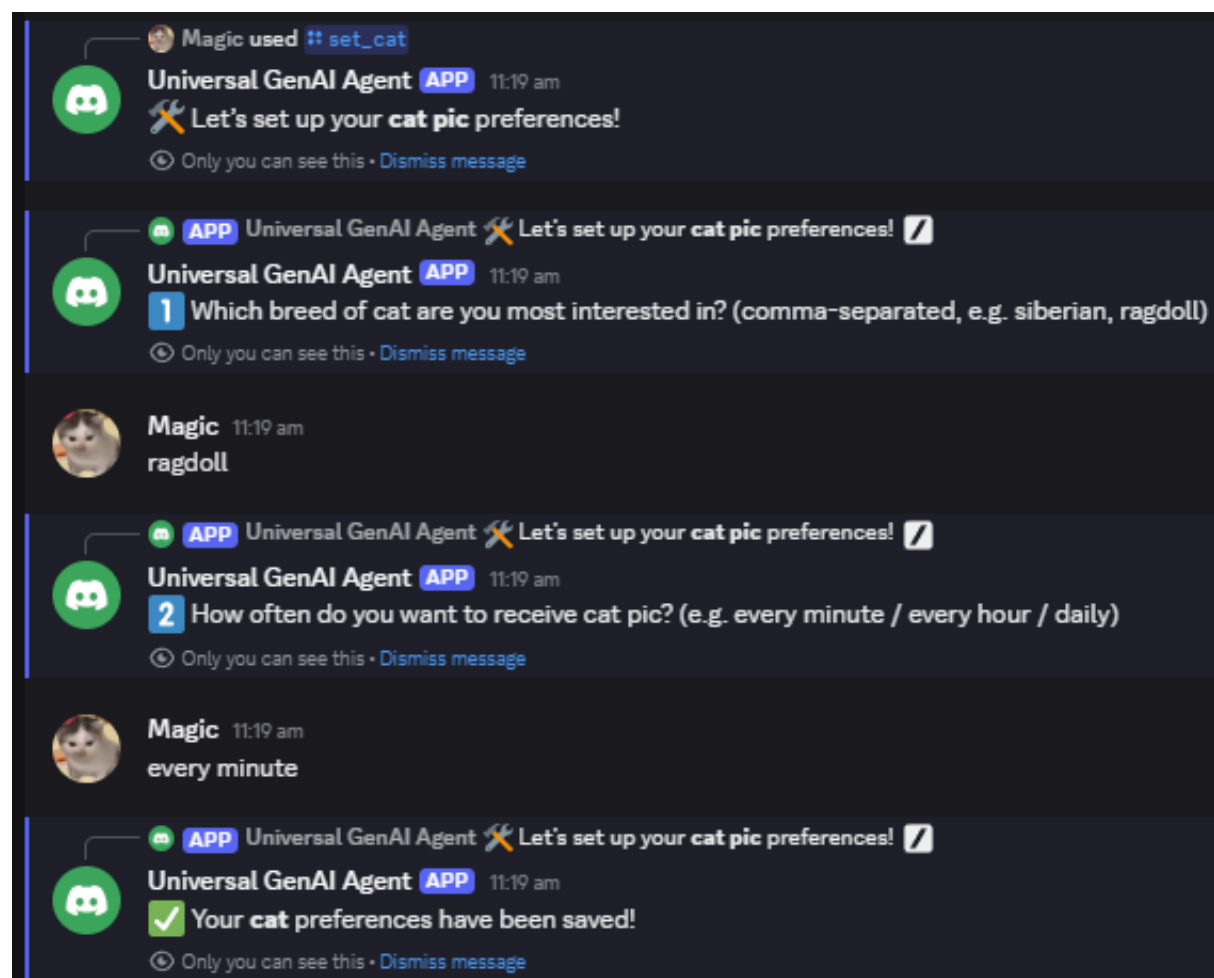


Figure 10: Example question and answering interaction of the `/set_cat` command that sets cat preferences of ragdoll breed and cron job frequency of delivering a cat picture every minute.

The cron job is set to execute every minute at 11:19am as shown in Figure 10. Figure 11 below shows the cron job delivering cat pictures every minute.

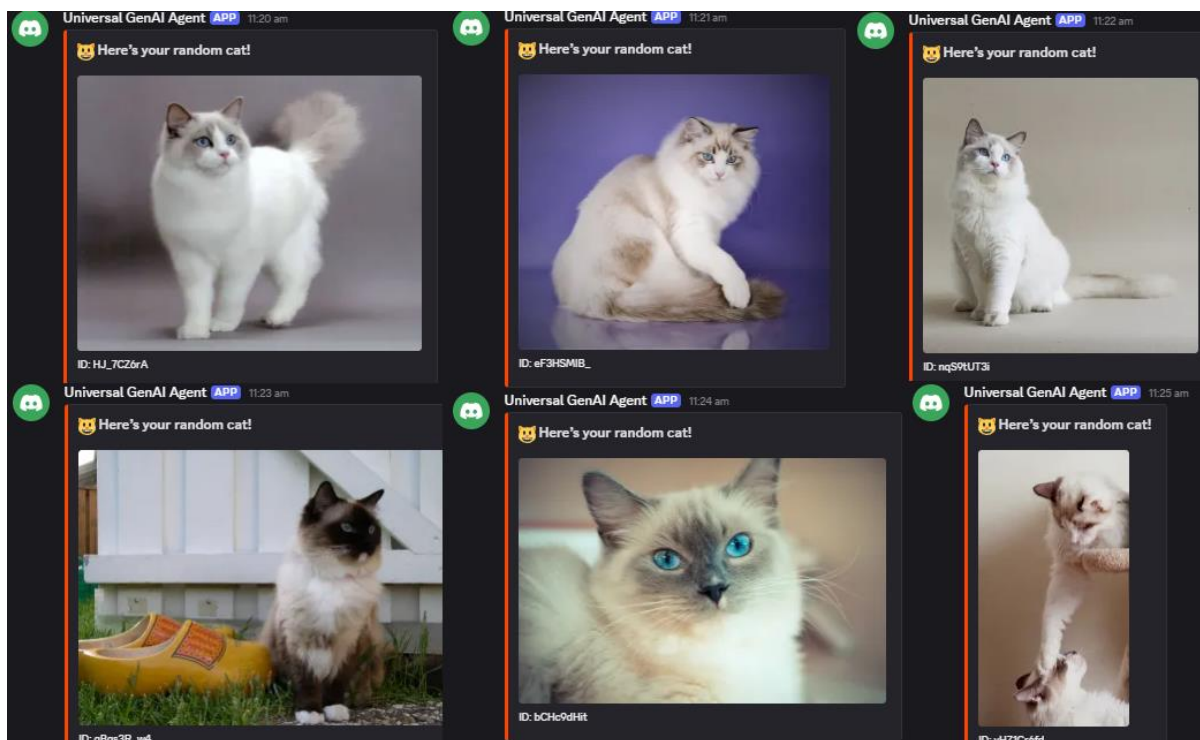


Figure 11: Example output from the cron job every minute from 11:20am to 11:25am according to user preference set in Figure 9 (ragdoll breed, deliver every minute).

This personal interest tracking functionality is not limited to cat pictures. By the easy integration nature of the Universal GenAI Agent, the implementation only requires a new API connector that retrieves content to the Master Server. Given the abundance of APIs nowadays for almost every usage, a very broad range of applications can be supported. Similar commands can be designed for other interests, such as /meme for retrieving favorite memes or /quote for daily motivational quotes to give the user a good mood. By allowing users to set preferences for various content types, the Universal GenAI Agent creates a more engaging and tailored experience across different areas of interest.

Overall, this approach to personal interest tracking enhances user interaction and satisfaction, making the Universal GenAI Agent a versatile tool for accessing content that resonates with individual users.

3.3 Web Source Content Retrieval

In some specific or niche cases, there may not be an API for the retrieval of desired content. In such cases, an active content extraction tool will be required. The system's Selenium web scraping module is designed to handle such cases. The capability of active content retrieval from any web source significantly enhances the flexibility of the Universal GenAI Agent and allows it to handle a much wider range of use cases.

- Command /analyze: Analyzes content from any website and answers user's follow-up questions.

The following figures demonstrate the functionality of the /analyze command. Figure 12 shows how to execute the command using the URL “scmp.com”, which is a news website. Figure 13 presents the response generated from this analysis, while Figure 14 demonstrates the interaction potential and illustrates the response to the user's selected follow-up question.

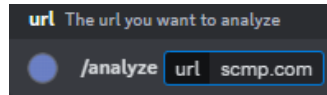


Figure 12: Guide to execute the /analyze command, by typing /analyze [url] and using “scmp.com” in the url field as an example website to scrape content

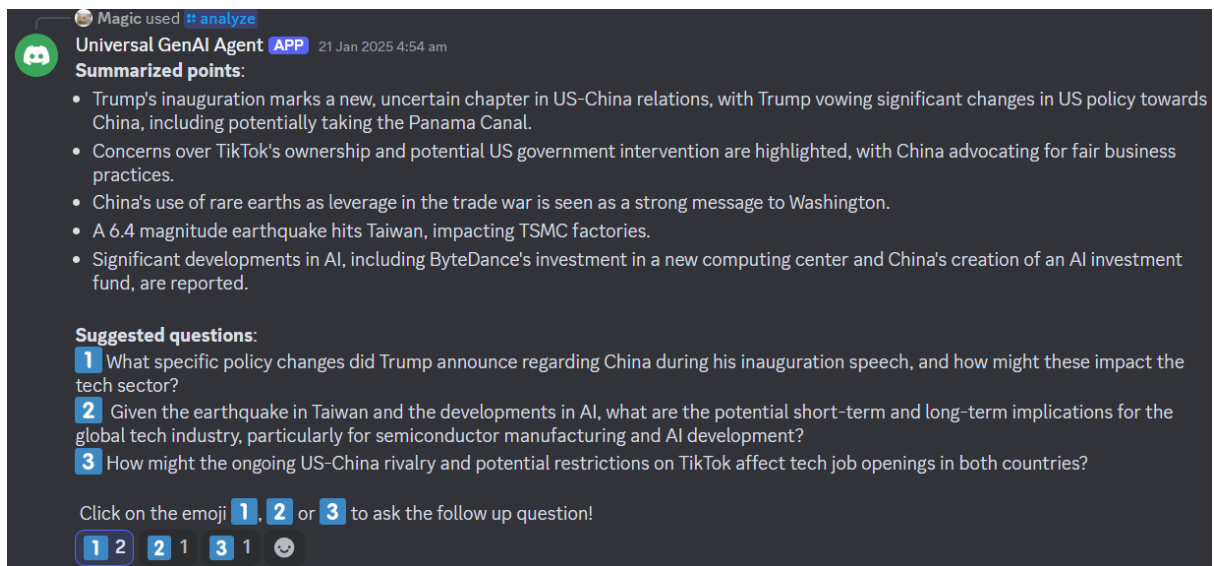


Figure 13: Example response after the command /analyze has been executed with url “scmp.com”

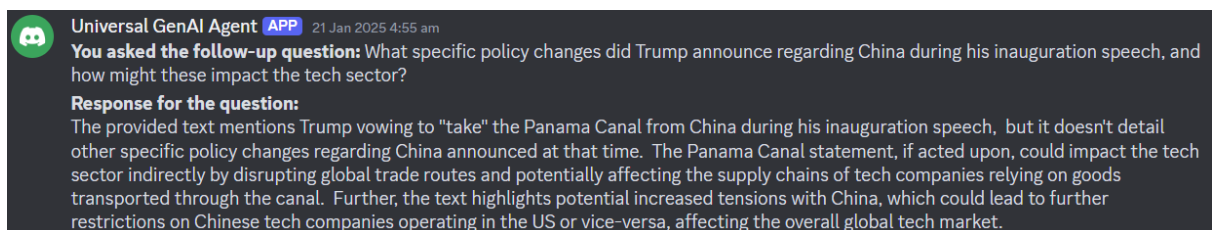


Figure 14: Example response after asking a follow-up question to the previous response of /analyze

To achieve a higher level of customization to cater to different users' preferences, command `/set_analyze` can allow scheduled delivery of a user's selected content that has been analyzed and personalized to match the user's interested topics.

- Command `/set_analyze`: Set the source, topics and delivery frequency of cron job.

Figure 15 below demonstrates how to execute the command `/set_analyze`. Figure 16 below demonstrates the interactive steps to set user's custom source to retrieve content and analyze and the cron job frequency.

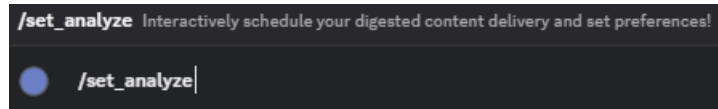


Figure 15: Guide to execute the `/set_analyze` command, input `/set_analyze` to set analysis preferences.

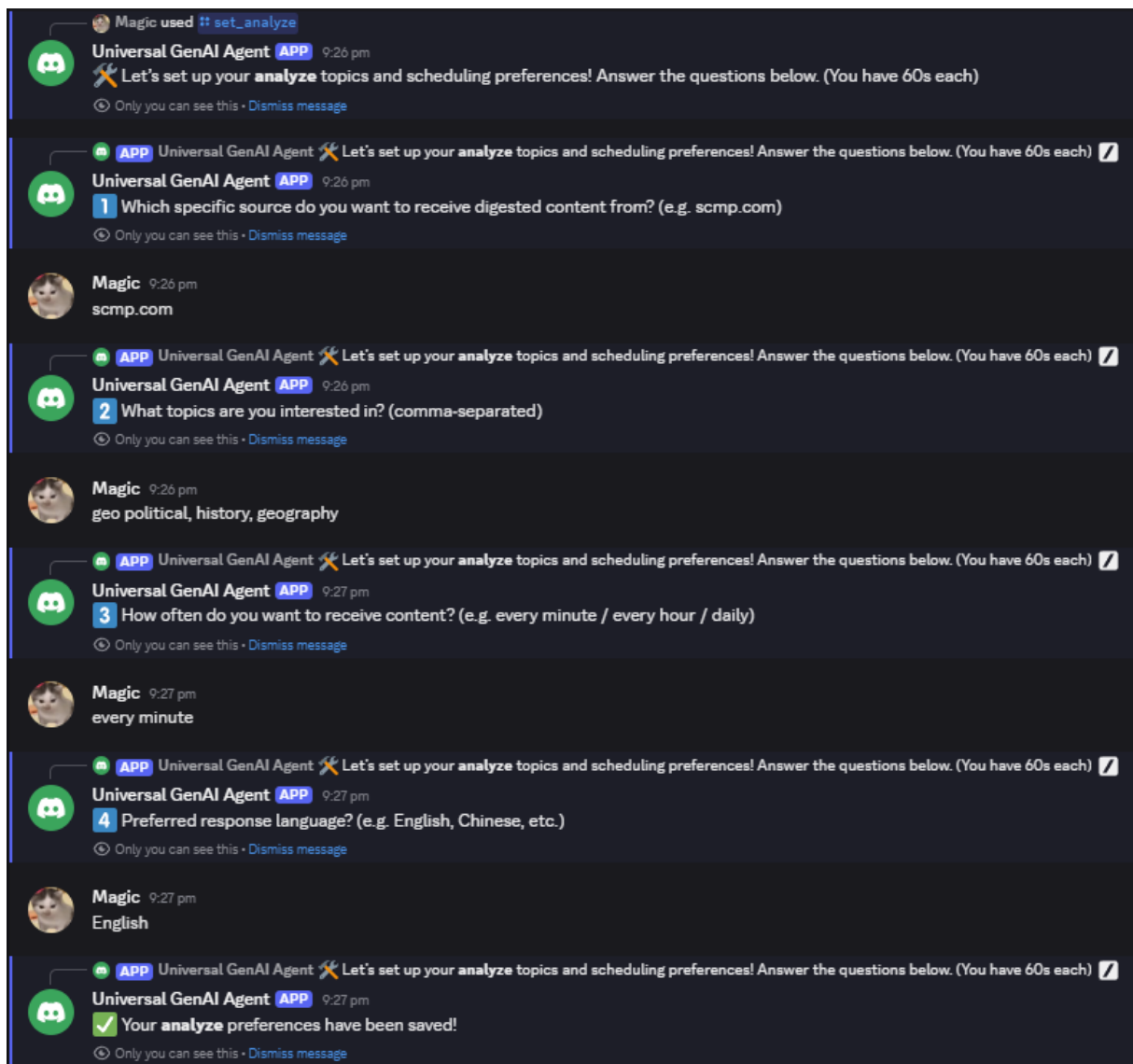


Figure 16: Example question and answering interaction of the `/set_analyze` command that sets user's custom web source to retrieve content for automated digested delivery (e.g. `scmp.com`), the user's topics of interests (e.g. `geo-political, history, geography`), and the cron job frequency (e.g. `every minute`).

Figure 17 below shows the cron job for /analyze command delivering the analyzed content.

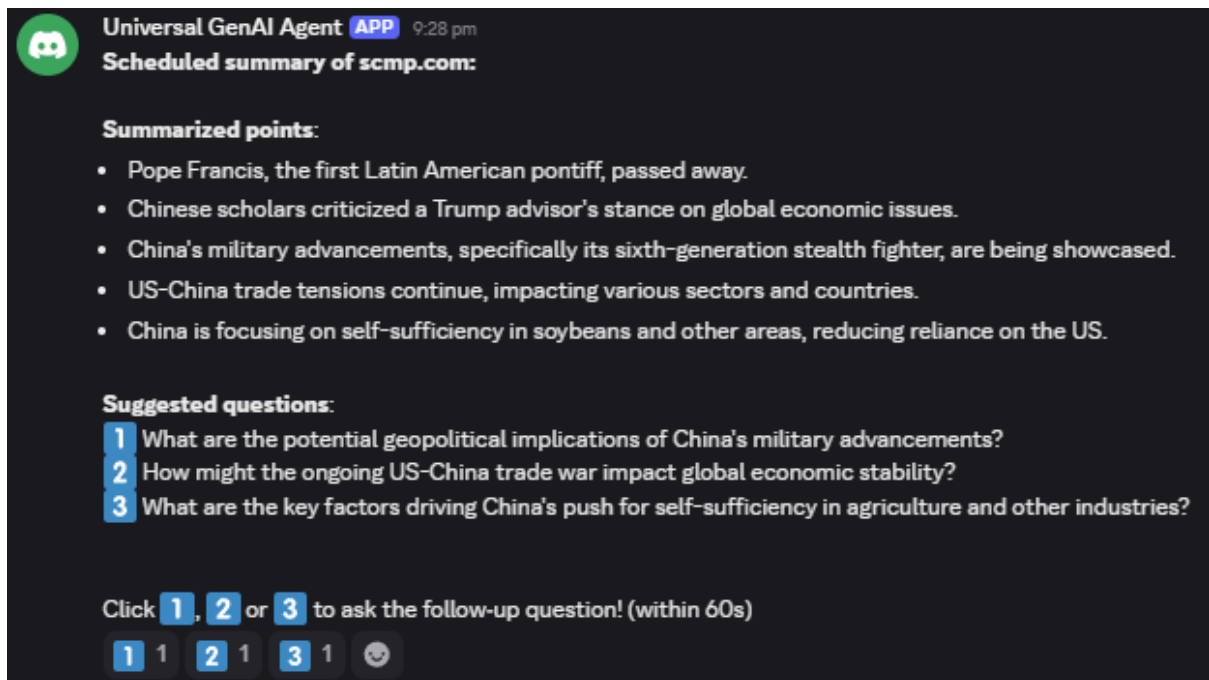


Figure 17: Example output from the cron job of /analyze according to user preference set in Figure 16. The content has been personalized to the user's set preferences, with most of the summarized content and suggested questions to ask being related to the geo-political topic that matches the user's interest.

The full system flow, from Selenium web scraping module and retrieving user's stored preferences in the database, to querying the Gemini GenAI service, has shown promising performance. Tests on 50 websites indicate that full content retrieval and analysis process consistently complete in under 15 seconds. Take financial news as an example, the system allows timely access to the latest job openings and easy-to-read descriptions, effectively reducing information overload and cognitive effort, showcasing the potential benefits brought by the real-time capability of the Universal GenAI Agent.

3.4 Clear Stored Preferences

- Command /clear_preferences

Allows users to clear their stored preferences in case they no longer needed the cron jobs.

The matching record with the user ID will be completely removed from the database and all cron jobs for the user will be stopped. Figure 18 below demonstrates how to execute the command /clear_preferences.

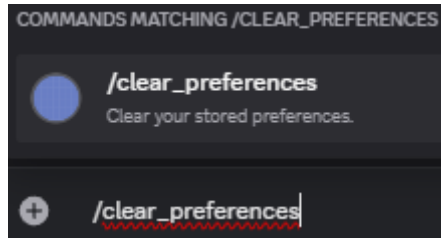


Figure 18: Guide to execute the /clear_preferences command, by typing /clear_preferences in the chat.

This command allows users who do not wish to retain any content for privacy concerns and ensures compliance with data protection regulations that users can always request to delete their stored data. Figure 19 below shows the output after executing /clear_preferences: a confirmation to a user after the user's data and preferences in the database have been deleted.

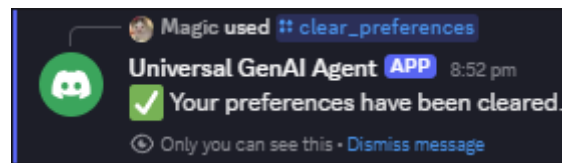


Figure 19: The confirmation sent to user after successful removal of the user's data.

4. Final Progress and Future Enhancement Ideas

The Universal GenAI Agent project has progressed according to the timeline in Table 2 below:

Date	Task	Status
September 2024	<ul style="list-style-type: none"> - System Modules: <ul style="list-style-type: none"> ○ Selenium web scraping module ○ Gemini GenAI service connector 	Completed
October 2024	<ul style="list-style-type: none"> - System Modules: <ul style="list-style-type: none"> ○ Discord API setup - Course Deliverables: <ul style="list-style-type: none"> ○ Project Website ○ Detailed Project Plan 	Completed
November 2024	<ul style="list-style-type: none"> - System Modules: <ul style="list-style-type: none"> ○ Discord commands: /analyze 	Completed
December 2024	<ul style="list-style-type: none"> - System Modules: <ul style="list-style-type: none"> ○ Third party API connector ○ Discord commands: /jobs 	Completed
January 2025	<ul style="list-style-type: none"> - Course Deliverables: <ul style="list-style-type: none"> ○ First Presentation ○ Interim Report 	Completed
February 2025	<ul style="list-style-type: none"> - System Modules: <ul style="list-style-type: none"> ○ NeDB Database Setup 	Completed
March 2025	<ul style="list-style-type: none"> - System Modules: <ul style="list-style-type: none"> ○ Cron Job Setup ○ Discord commands: <ul style="list-style-type: none"> /set_analyze, /set_jobs, /cat, /set_cat, /clear_preferences - Preliminary Implementation and Testing 	Completed
April 2025	<ul style="list-style-type: none"> - Performance Evaluation - Final Implementation - Course Deliverables: <ul style="list-style-type: none"> ○ Final Presentation ○ Final Report 	Completed

- Future Enhancement Ideas:
 - /sports: Provides live updates or scores for ongoing games in selected sports (e.g., football, basketball).
 - /flights: Checks for cheap flight deals and travel alerts.
 - /music: Provides recommendations for songs or playlists based on user mood.
 - /events: Lists upcoming local events or concerts based on user interests.
 - /meme: Lets users create and share customized memes using popular templates or their own images, complete with personalized captions for humor and creativity.
 - /quote: Offers users a selection of inspirational and motivational quotes, allowing them to browse or receive random quotes based on themes, and even submit their favorites.

These common usages of data delivery can be achieved the same way as /cat by integrating third-party APIs into the proposed enhancements will significantly elevate user experience across various features. By integrating live sports updates, flight alerts, personalized music recommendations, local event listings, meme creation, and inspirational quotes, users will enjoy a rich and engaging platform. These features collectively provide real-time information and tailored content, fostering creativity, community involvement, and a personalized experience that caters to diverse interests and needs. This approach not only fosters engagement but also streamlines access to relevant data, making the platform more interactive and user-friendly.

5. Conclusion

The Universal GenAI Agent project offers an innovative solution to the challenge of information overload. By integrating real-time web scraping, advanced Generative AI (GenAI), and personalized content delivery, the system enables users to access meaningful, actionable insights in a fraction of the time required for manual data retrieval and processing. This innovation promotes convenience, addressing cognitive fatigue and inefficiencies in information management, and fundamentally redefines how users engage with the digital world.

The system's modular architecture — comprising the Discord Chat Interface, Selenium Web Scraper, API connectors, NeDB, and Gemini GenAI — allows high adaptability in fine tuning to support diverse use cases. From job seekers seeking timely updates to individuals tracking personal interests, the Universal GenAI Agent delivers personalized, real-time content tailored to unique user needs, and demonstrates substantial reductions in data processing time and cognitive effort, highlighting its potential to enhance user productivity and decision-making.

Ultimately, the Universal GenAI Agent represents an attempt in the application of GenAI for personalized content delivery. By addressing critical gaps in existing information retrieval systems, this project generates personalized and contextually relevant outputs that dynamically adapt to evolving user preferences. Its potential applications extend well beyond the realms of job searching and personal interests monitoring; they encompass diverse areas such as social media trend analysis, health monitoring alerts, and personalized educational content delivery.

In conclusion, the Universal GenAI Agent demonstrates the potential of GenAI technologies in optimizing workflows and enhancing decision-making across various sectors. It addresses the challenges of information overload, investigating AI's ability in creating a more efficient and personalized digital experience for users. This foundational work not only paves the way for future innovations in AI-driven content delivery systems, but also highlights the crucial role of AI in reshaping how we manage, interact with, and derive value from information.

References

- [1] J. Spira, "Information Overload: A Systematic Review of the Literature," Information Overload Research Group, 2019.
https://www.researchgate.net/publication/265906917_Information_Overload_A_Systematic_Literature_Review (accessed Oct. 14, 2024).
- [2] S. Ono, "Unlock the information advantage to combat 'information overload'," OpenText Blogs, 2022. <https://blogs.opentext.com/unlock-the-information-advantage-to-combat-information-overload/> (accessed Nov. 30, 2024).
- [3] E. O. Sodiya, O. O. Amoo, U. J. Umoga, and A. Atadoga, "AI-driven personalization in web content delivery: A comparative study of user engagement in the USA and the UK," *World Journal of Advanced Research and Reviews*, vol. 21, no. 2, pp. 887–902, 2024. <https://doi.org/10.30574/wjarr.2024.21.2.0502> (accessed Oct. 14, 2024).
- [4] B. Smith, A. Johnson, and R. Lee, "Exploring the Benefits of AI for Content Retrieval," *Journal of Information Science*, vol. 45, no. 1, pp. 1-10, 2023.
https://www.researchgate.net/publication/378575962_Exploring_the_Benefits_of_AI_for_Content_Retrieval (accessed Oct. 14, 2024).
- [5] "API docs for bots and developers," Discord Developer Portal,
<https://discord.com/developers/docs/interactions/receiving-and-responding#receiving-an-interaction> (accessed Nov. 30, 2024).
- [6] "Node.js - about node.js," Node.js, <https://nodejs.org/en/about> (accessed Nov. 30, 2024).
- [7] "OpenAI Platform," OpenAI, 2024. <https://platform.openai.com/docs/guides/rate-limits/usage-tiers?context=tier-free> (accessed Oct. 16, 2024).
- [8] "Gemini API Pricing | Google AI for Developers," Google AI for Developers, 2024.
https://ai.google.dev/pricing#1_5flash (accessed Oct. 16, 2024).