University of Melbourne

A Cloud-Native System for Social Media Mining of AI Topics in Australia

Student :
Junbo Liang 1019905
Yutao Zhou 1001087
Yuyao Zhang 1509178
Louis Liu 1659592
Liyang Wang 1543938

Teacher :
Richard Sinnott
Luca Morandini

May 21, 2025

Contents

1 Introduction

In recent years, artificial intelligence (AI) has developed rapidly. It is not only active in the field of chatbots, but applications such as AI drawing are also very popular in improving people's work efficiency. To understand public attitudes and concerns about AI-related topics, we decided to track how AI is involved in public life and the sentiments of various communities towards these topics.
The project aims to develop a scalable cloud-based system that can collect and standardize data, extract and analyse data, and a widget-rich user interface to control the system and display analysis results. We use Fission for real-time stream processing and Elastic Search to store data, automatically collecting data from social media platforms such as Mastodon and Reddit. We use Jupyter Notebook to implement a flexible front-end interface. Through this system, we investigate four analytical scenarios including user activity patterns, sentiment trends, daily post analysis, and topic correlations.
This report begins with a brief introduction to the system architecture and the responsibilities of the core components, then goes into depth on the design and implementation of each key component in the system, and describes the deployment of the ReCTAR Research Cloud. Followed by the evaluation on analysis results, and the pros and cons of technology choices such as the NeCTAR Research Cloud with Kubernetes, as well as the pros and cons of using Fission, and ElasticSearch for big data analysis. Then the videos demonstration of the system, the challenges we encountered and our solutions, and finally the conclusion and future work.

2 System Architecture

2.1 System Architecture Diagram

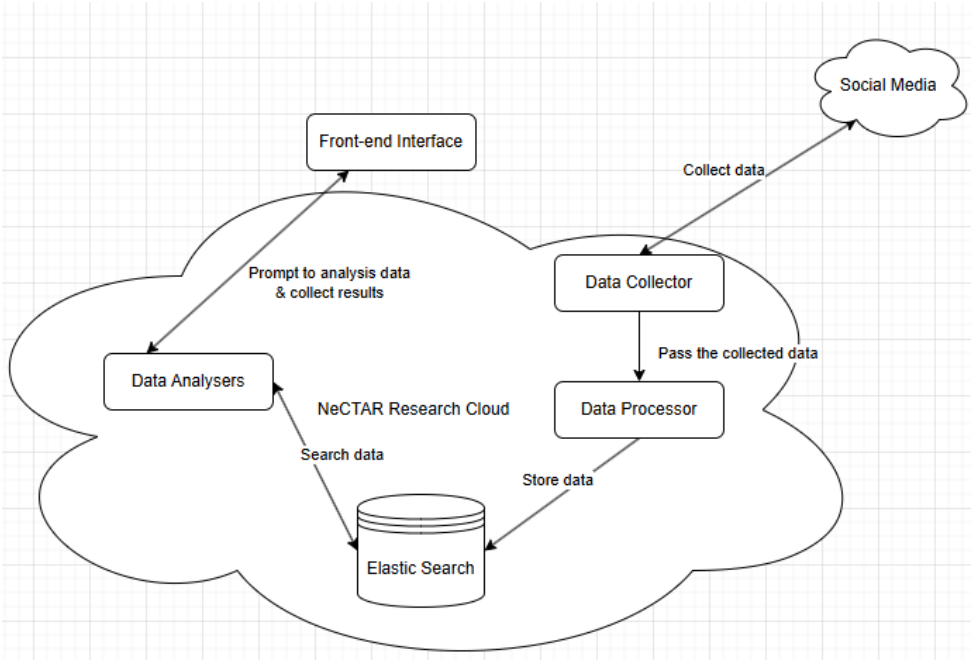

Figure 1: System Architecture

Our system has six main components, here is their descriptions and purposes:

- Front-end interface: Written in python and runs on Jupyter Notebook. It allows the user to interact with the front-end widgets and displays output. It is deployed in the cloud but allows users to connect remotely, execute the code, and map the notebook's graphical interface to their own machines.
- Data Collector: Written in Python and deployed in the cloud. The data collector receives commands, given tags, and social media APIs from the front-end program, then collects data from social media and stores it in Elastic Search.
- Data Processor: Pipeline rules that normalize collected data before storing it in elastic storage.
- Elastic Search: It is a search engine that acts as a database in our system. It allows data collector and data processor to write data and allows data analyser and data processor to extract data from it.
- Data Analysers: Written in python and deployed in the Cloud. It receives commands from the front-end program, extracts data from Elastic Search and performs calculations, and then returns the required data to the front-end program.
- NeCTAR Research Cloud: The research cloud organized by the Australian Research Data Commons deploys the above four components.

2.2 Programming Languages

In our software, the front-end and back-end programs are mainly written in Python, which is a language that every member is familiar with, easy to write and has enough available packages. And use yaml for environment configuration and deployment.

2.3 Data Collector Components

In our software architecture, we designed several data collectors to fetch posts about Australian AI from different platforms in the past few years and put them directly into the elasticsearch index. Due to the large number of posts, we cannot c all the past posts at once. Therefore, each time the data collector is called, the last_id of the post will be saved to facilitate the location of the post next time. The data collector is deployed to K8s on Nectar Research Cloud through fission, and a timer is set to periodically fetch posts.

2.4 Data Processor Components

We implemented a set of complete data process components which are used for cleaning and standardizing the original data collected from social media platforms. These components are divided by functions, including array conversion, content cleaning, HTML removing, punctuation marks processing, space standardization and AI content filtering, etc. Each component is responsible for a specific part in the data processing flow, and these components work together to constitute an efficient data processing pipeline.

2.5 Elastic Search Components

We created several different indexes in elasticsearch through curl prompts to store data of different platforms and topics. We have built a flexible data processing system by using the ingest pipeline feature of Elasticsearch. By defining multiple specialized processing pipelines, we can transform, clean and filter raw data from different platforms. The Elasticsearch pipelines use Painless scripts to implement complex text processing logic, including standardization, content filtering and specific topic classification. We have created multiple dedicated indexes to store data from different platforms and topics, facilitating subsequent analysis and visualization. This approach eliminates the need for additional data processing infrastructure and simplifies the entire data processing workflow.

2.6 Data Analysers Components

According to the software architecture, the data analysis program needs to respond to the front-end request and return the corresponding analysis results. Therefore, there is a scheduler program as a bridge between the front-end program and the back-end functions, which is responsible for analysing the information sent by the front-end and running the corresponding functions.
And there are four data analysis functions in back-end, each corresponding to a research topic, which will be introduced in more detail in the following sections.

2.7 Front-end Components

The front-end is written in Jupyter Notebook, which is easy to implement and displays the output directly. It is more flexible to use for users who know Python but not other front-end languages such as React. They can write additional code while the interactive UI is running to get the desired output, which makes customization easier. For users who are not familiar with programming languages, they can load our Notebook and run all the function blocks in order and then get the information they want through the interactive widgets at the bottom of the Notebook.
In our software architecture, the front-end is responsible for requesting the data collection program to collect data, requesting the back-end program to clean and analyse the collected data, and receiving and displaying the results.
We use widgets from the ipywidgets package for interaction. To ensure that the widget works properly, nbextentions needs to be enabled. And the version of Jupyter Notebook needs to be lower than 7.0.0, because the latest version of Jupyter Notebook no longer supports the nbextentions plugin.
We also use the package of matplotlib for displaying our analysed result. In our system, we used classic charts such as bar charts, line charts, and pie charts, as well as the word cloud charts from the wordcloud package.

2.8 Front-end & Back-end Communication

We use Flask for front-end and back-end communication. Flask is a lightweight Python microframework for building web applications, which is a good fit for our software. According to our design architecture, the amount of data transferred between the front-end and back-end is very small and the frequency is low, and no complex communication

framework is required, so using the easy-to-implement Flask is very beneficial. In addition, it is very suitable for building RESTful API, which is an interface that communicates through the HTTP protocol. Therefore, the front-end can send requests to the back-end programs through API access.

2.9 Scalability

In our system, each key component is relatively independent, communicating and decoupling through HTTP API, which is conducive to efficient horizontal expansion. The entire system is deployed and run in the cloud and can be horizontally expanded by applying for more resources. And uses Elastic Search as a search engine and a database that allows automatic data replication and sharding, which can dynamically manage the storage of data as more data is collected.
If there is new interest to be researched, it can be achieved by updated separately both the backend logics and add more features to the front-end. The front-end interface has enough widgets and multiple different ways to visualize the results and can send different requests to get relevant data from the back-end and display it. The front-end interface can also prompt the data collector to run automated scripts to collect the required data from social media through a given authenticated API. For more information on scalability, see the video demo in the Upcoming Demos section.

3 Design Details

3.1 Data collectors (Mastodon/Reddit clients)

Since our topic is related to Australian AI, we wrote an automated data collector called `mastodon_ai.py` in Python, with the goal of continuously scraping posts related to artificial intelligence (AI) from the `https://mastodon.au/`. Our collector can not only collect data, but also directly pass the collected data into the elasticsearch index. I will explain the implementation logic of the collector in great detail.
1. Use Mastodon (api_base_url=`'https://mastodon.au'`) to establish a connection with mastodon.
2. Elasticsearch in this project is deployed in the elastic namespace of Kubernetes, and the Service name is elasticsearch-master. Therefore, we can access the Elasticsearch service from the Kubernetes cluster through the following internal DNS address: `https://elasticsearch-master.elastic.svc.cluster.local:9200`. In the actual code, we create a client connection in the following way: Elasticsearch(`https://elasticsearch-master.elastic.svc.cluster.local:9200`). This address is only valid within the Kubernetes cluster and is applicable to Fission functions or other components running in Pods, realizing service discovery and communication between containers.
3. Use the ‘timeline_hashtag‘ interface to capture posts with the ‘#AI‘ tag. Due to the limitations of mastodon itself, only 40 posts can be fetched each time. Use ‘max_id‘ to implement the **from new to old** archiving mode, obtain historical data in pages, and expand to 500 posts each time.
4. Use the ’extract_main_info()’ function to extract the key information of each post, including post id, release time, content, author username, tag list, and image link.

5. Use an independent metadata index ‘mastodon_ai_last_id‘ to save the ID of the oldest post in this round (‘max_id‘) after each fetch to achieve breakpoint continuation.
6. The system uses the standard ‘logging‘ module to output the number of pages, total number, and failure information for easy debugging and monitoring.
7. Use ’id’ as the primary key field in Elasticsearch. If the same id is written, the original content will be overwritten to avoid duplicate writing.
This collector can be easily integrated with serverless platforms such as Fission to continuously capture past data through timed triggers.
In addition to mastodon data collector, we also wrote an automated data collector called ’reddit_public.py’ to collect posts from different regions in Australia. I will explain the implementation logic of the collector in great detail.
1. Use praw.Reddit(..) to establish a connection with reddit.
2. Use reddit.subreddit() to crawl different sections, such as Melbourne, Sydney and so on.
3. Like mastodon, extract the key information of each post.
4. Set time.sleep(1) to prevent triggering Reddit’s speed limit (HTTP 429)
The rest is the same as mastodon data collector. Due to the limit of reddit api, we cannot directly crawl posts about AI in Australia on reddit, so the data collector we designed for reddit only crawls posts from various parts of Australia and transmits them to elasticsearch, and then filters posts about AI through data cleaning.

3.2 Fission for stream data processing and data ingestion

In order to realize the automatic execution and in-cluster management of the collector, we use the Fission serverless computing platform to deploy the data fetching script as a function service. Taking the ‘mastodon_ai‘ data collector as an example, its file structure includes: mastodon_ai.py, requirements.txt and build.sh. mastodon_ai.py is the main fecthing post logic, and requirements.txt contains the dependencies required by mastodon_ai.py, such as elasticsearch8 and mastodon.py. The build.sh is a script for building function packages and uploading them to Fission. The function is created and deployed using the following command:

```
fission package create --name addmastodonai \
  --sourcearchive ./fission/functions/mastodonai.zip \
  --env python --buildcmd './build.sh'
fission fn update --name addmastodonai \
    --pkg addmastodonai --env python  \
    --entrypoint "mastodon_ai.main" \
    --fntimeout 3600
```

To schedule the job periodically, we use:

```
fission timer create --name mastodonaitimer \
  --function addmastodonai --cron "@every 130s"
```

Because the topic we are studying is about posts from AI in Australia in recent years, the role of mastodonaitimer is to continuously call the mastodon data collector to continuously collect posts about AI in Australia in the past. So far, the collection has reached 2023, and the number of collected posts has reached more than 130,000.

3.3 Data storage layer (ES index and mapping)

This project uses Elasticsearch 8 as the main structured data storage engine. All collected Mastodon posts about AI will be written to an index named 'mastodon_ai'. The mapping configuration of the index is shown below:

Listing 1: Elasticsearch Index Mapping

```
{
  "mappings": {
    "properties": {
      "id":                 { "type": "keyword" },
      "created_at":         { "type": "date" },
      "url":                { "type": "keyword" },
      "content":            { "type": "text" },
      "author_username":    { "type": "keyword" },
      "author_display_name": { "type": "text" },
      "tags":               { "type": "keyword" },
      "media_urls":         { "type": "keyword" }
    }
  }
}
```

The writing strategy is to use id as the primary key for each data, and use `es_client.index(index="mastodon_ai",id=post["id"],body=post)` to write to elasticsearch index. If id exists, content will be overwritten to avoid duplicate writing. After each round of fetching, last_id will be written to mastodon_ai_last_id index for the next fetching. In order to ensure high availability and later scalability of data, we set the following parameters when creating the index:

Listing 2: Elasticsearch Index setting

```
    "settings": {
      "index": {
        "number_of_shards": 3,
        "number_of_replicas": 1
      }
    }
```

Setting number_of_shards to 3 means splitting the index data into three primary shards, supporting concurrent reading and writing of data and improving the efficiency of reading and writing. Setting number_of_replicas to 1 means retaining a replica for each primary shard, ensuring that when the primary shard fails, the replica shard will automatically become the new primary shard, thus ensuring data availability and security.

3.4 Front-end Interactive widgets

In the UI of this system, we used text box widgets, single and multiple selection widgets, check box widgets, date picker widgets, and button widgets, and displayed clear names of each widget on the UI so that users can understand the functions of these widgets. This feature further simplifies our UI by only showing the widgets needed for each different research scenario, ensuring that no redundant information is displayed in the UI. For

example, when the user selects Topic 4 (analysing sentiment towards areas of interest over a given time period), the UI refreshes and displays a text box widget for the user to enter their areas of interest, whereas this text box does not exist when the user selects other scenarios for analysis.
There is a function of automatically scheduling chart drawing, executing different chart drawing functions according to the current research topic and the selected chart type, and providing information such as title name for the drawing function to customize the drawn chart.
There is also a feature that dynamically updates the widget's value when the user selects a different theme to provide a default combination of settings to analyse and display the data. It can limit the option selection of the rest of the widgets to prevent potential errors, such as Topic 1 (Find the most active users), the front end will not receive enough information to display charts other than line charts, so in the multi-select widget for chart type selection, it will only show the line chart option.
There is also a function to clip data based on a given time period, allowing the user to adjust the display range of the line chart. For example, the user may want to zoom in on a certain portion of a line chart, which he can do by adjusting the 'start_time' and 'end_time' date picker widgets and clicking on the display again, thereby drawing a new chart that zooms in on that specific portion of the chart.

3.5 Data Processing

Since the Reddit data contains all types of topics and content from Reddit, and our research specifically focuses on discussions about artificial intelligence, we needed to implement a filtering pipeline to exclude the content unrelated to AI. We developed a data processing pipeline to filter out AI-related discussions from the raw data obtained from Reddit. The core of this pipeline is a precise string matching mechanism implemented using Painless scripts, which can identify "AI", "ai", "A.I." as standalone words while excluding cases where they are part of other words (such as "paid" or "said").
We first converted the array fields of Reddit (such as author, content, title, etc.) into single values to ensure consistency then we created a cleaned_content field which come from original content by a series of standardization processes, including converting to lowercase, removing HTML tags and punctuation and normalizing whitespace characters. These steps made the cleaned_content more suitable for analysis while retaining the original content field for reference in its original format when necessary.
Our filtering pipeline reduced the data volume, retaining only the content truly related to AI discussions. Through this approach, we were able to precisely locate discussions about artificial intelligence within the vast Reddit content, regardless of whether they are about AI or general topics.
The metadata of Reddit, such as author information (author), posting time (created_utc), sub-forum (subreddit), score (score), and number of comments (num_comments), were also retained, enabling us to conduct time series analysis and study community participation patterns. Although the filtered reddit_basic_ai_only index only contained content related to the AI theme, the filtered data volume was too small to be use in analysis, so we turned to Mastodon.
The data structure of the Mastodon platform differs from Reddit's, especially in terms of content format and user interaction methods. Mastodon content includes a lot HTML

elements, mention tags, and hashtag and other structured information. Our Mastodon processing pipeline is also focused on extracting and cleaning AI-related content, while preserving the platform's unique features. The processing involves converting arrays to single values (except tags field), creating a standardized cleaned_content field, removing HTML tags, and normalizing text formats. Unlike Reddit processing, we keep the tags field of Mastodon in array format to keep the user-defined topic classifications. This processing retains the unique tag field of the Mastodon data, allowing us to analyze how users categorize and organize AI-related content through tags. The federated structure characteristics of Mastodon, such as user interactions and references between different instances, are also preserved in the URL and content fields. The URL format retains the instance domain name and user identifier, while the content field retains the complete HTML structure of cross instance references. The cleaned_content field contains the processed content composed absolutely of words, which is not affected by symbols during analysis. The tag field in Mastodon data ensures that the extracted data is already related to AI, providing a large amount of data for our analysis after cleaning.

3.6 Data Analysis

The data analysis module consists of a scheduler and four functions corresponding to the each research scenarios, which will be introduced in the following section. There is a dictionary mapping front-end requests to corresponding analysis functions and they are executed according to the specifications provided by the request.
Below are details about the inputs, outputs, and packages used for each analysis function:

1. User Analysis and Feature Analysis (topn.py): Accepts a list of database names and a tag name as input; uses Counter to find the top 10 most active users and their number of posts, and uses DataFrame to find the number of posts per hour on weekdays and weekends, and analyses the target data according to the given database and tag. Returns the information of the top 10 users and a DataFrame of the number of posts per hour.

2. Sentiment Analysis on AI Topics (emotion.py): Accepts a list of database names and a tag name as input; uses SentimentIntensityAnalyzer from nltk which require the VADER lexicon to evaluate the sentiment score of each post, uses Counter to find the number of posts with positive, negative, and natural sentiment, uses Dataframe to store the daily average sentiment score, and analyses the target data according to the given database and tag. Returns the sentiment score distribution and a DataFrame of daily average sentiment score.

3. Heat Analysis (hotTopics.py): Accepts a list of database names as input; uses Dataframe to store the daily post on 'AI' tag and analyses the target data according to the given database. Returns a DataFrame of number of daily posts.

4. The Correlation between AI Topics and Other Topics (tag.py): Accepts a list of database names as input; uses a counter to find how often the 'AI' tag appears with other tags and analyses the target data according to the given database. Returns a dictionary of tag names and their corresponding frequencies.

3.7 Flask & Restful API

In the scheduling program, the flask package is imported and initialise a Flask application object via 'app = Flask(__name__)'. And use the '@app.route' decorator to declare a POST route '/api/run' to receive the JSON[1] request from the front-end.
Correspondingly, the front-end program requires a Restful API from the back-end to allow it to send requests to the back-end program. By default, Flask applications execute on localhost using port 5000, so the API is of the form `http://localhost:5000/api/run`. After clicking the "Search" button, a request containing information such as research topics, databases, etc. will be packaged into JSON and sent as a POST request to the backend through the API using the 'requests.post' function. Then receive the returned results and analyse and display them.
Since HTTP communication can fail for many reasons, such as the backend function being unable to connect to the Elastic Search server, it is always worthwhile and safe to use try and except for running communication-related code blocks.

4 Deployment

In this project, we mainly used docker to deploy our front-end and back-end codes. Docker is by far the most successful containerization technology, which container is a concept of resource isolation and allocation as a virtual machine without bundling the entire environment and full OS.

4.1 Backend

Begin by placing a Dockerfile in the root directory of the backend codebase, while specifies the necessary environment and dependencies. Utilize the RUN instruction to install all required packages listed in requirements.txt. Using the CMD instruction to execute our backend program entry point (dispatch.py) with Python. Build the Docker image locally and tag it as 'latest', then push the image to the Container Registry on our private Gitlab to make it accessible for the deployment later. Create deployment.yaml and service.yaml files to define the Kubernetes Deployment and Service resources. These files include the specifications such as the namespace (backend), resource requests and limits, the container image location (pointing to the GitLab registry), and the image pull secret for authentication. On the Kubernetes cluster hosted on the NeCTAR Research Cloud[3]., create a namespace named backend, and within this namespace, create an image pull secret using the GitLab username and access token to allow the cluster to authenticate and pull the container. Then use the local virtual machine running the ubuntu on WSL2, apply the deployment.yaml and service.yaml configurations to the cluster. This action deploys the backend application, creating the necessary pods and services within the backend namespace. After all these procedures, our backend program is deployed on cluster and available to be used.
Here are some of the key commands used in deploying the backend program:

Listing 3: backend deployment (Shell)

```
# Authentify with the gitlab username and access token
docker login registry.gitlab.unimelb.edu.au:5005
# Build the Docker image with tag latest (make sure current dir has Dockerfile)
```

```
docker build -t registry.gitlab.unimelb.edu.au:5005/yutazhou/comp90024_team_17:latest .
# Push the image to gitlab
docker push registry.gitlab.unimelb.edu.au:5005/yutazhou/comp90024_team_17:latest
# Deploy config (make sure current dir has this file)
kubectl apply -f deployment.yaml
# Service config
kubectl apply -f service.yaml
```

For the code of deployment.yaml, service.yaml, Dockerfile, please refer to our Gitlab.

4.2 Front-end

The deployment of the front-end Jupyter Notebook is basically the same as the back-end deployment described above, but there are slight differences in the details. It required Dockerfile, requirements.txt, both deployment and service .yaml file, the image pull secure and the private Gitlab for storing the image. The first difference is that in the Dockerfile, we need to use the RUN instruction to run jupyter_contrib_nbextentions and install it to enable the functionality of the front-end user interface widget, and we also need to define more specifications to make the notebook run in the mode we want.

1. "–ip=0.0.0.0": Listen on all available network interfaces and allowing external access.
2. "–port=8888": Specify port 8888 for the Jupyter Notebook.
3. "–allow-root": Allow to run the Jupyter Notebook as root user.
4. "–no-browser": Ban running browser automatically to prevent error running on the cluster.
5. "–NotebookApp.token=”" & "–NotebookApp.password=”": Ban the authentication, simply to access.

Secondly, the naming different of the image on Gitlab container registry, the namespace would be in frontend which is separated from the backend. Here are some of the key commands used in deploying the backend program:

Listing 4: front-end deployment (Shell)

```
# Authentify with the gitlab username and access token
docker login registry.gitlab.unimelb.edu.au:5005
# Build the Docker image with tag latest (make sure current dir has Dockerfile)
docker build -t registry.gitlab.unimelb.edu.au:5005/yutazhou/comp90024_team_17:frontend .
# Push the image to gitlab
docker push registry.gitlab.unimelb.edu.au:5005/yutazhou/comp90024_team_17:frontend
# Deploy config (make sure current dir has this file)
kubectl apply -f deployment.yaml
# Service config
kubectl apply -f service.yaml
# Get the name of pod
kubectl get pods -n frontend
# Port-forward to allow access on local machine
kubectl port-forward deployment-86f5554cb-c797l 8888:8888 -n frontend
# Open the notebook on browser
http://localhost:8888/
```

Since the front-end program is the entrance to the entire system, it can control the back-end program to perform data analysis and display the results. Therefore, we need to perform port forwarding from cluster to local machine to access the back-end program.

4.3 Kubernetes Cluster,Elasticsearch and Fission

First, I deployed the Kubernetes cluster to the cloud (NeCTAR Cloud) using the **openstack coe cluster create** command, and set up three worker nodes and a master node. Then I installed ElasticSearch into the current cluster using Helm and set up a namespace "elastic". Fission is similar to ElasticSearch. All fission components are created in the cluster using the Helm command and the namespace "fission" is set up.

5 Evaluation & Analysis

The Scenarios selected for our system:

1. User analysis and feature analysis of the Mastodon platform
2. Sentiment analysis on AI topics
3. Heat analysis
4. The correlation between AI topics and other topics

5.1 User analysis and feature analysis of the mastodon platform

5.1.1 Scenario description

The code focuses on analyzing the posts related to AI posted by Australian users on the Mastodon platform, identifying the most active users by counting the number of posts posted by users, and analyzing the keyword distribution of the topics they participate in, so as to identify potential "opinion leaders" or topic promoters.

5.1.2 Why we choose this

Australian users are increasingly active in discussions on AI applications, education, and policies. As a multicultural country, the region's users are representative in terms of topic participation and focus, and are worthy of further study.

5.1.3 User image analysis

As shown in Figure 2, we aggregated the posting data on the platform, filtered out the users with the most posts, and extracted their common nicknames and popular tags. From the results, user `@androiddreams` (Android Dreams) is currently the most active AI content publisher, with a total of 26,596 posts, far exceeding other users. Further observing the top 3 tags used by these "high-frequency speakers", we found some typical content tendencies:

- `@androiddreams`: Focused on `stablediffusion`, `aiart`, and `aigenerated`, showing deep involvement in AI visual creation.
- `@itnewsbot`: Tags such as `machinelearning`, `biz`, and `ai` indicate a focus on technology news.

- `@redhotcyber`: Uses `redhotcyber`, `hacking`, and `ai`, reflecting attention to AI security and hacker domains.
- `@alvinashcraft`: Mixed tags like `dotnet`, `cloud`, and `ai`, representing a developer perspective.

Some nicknames also reflect strong "information source" characteristics, such as *Crypto News*, *Bytes Europe*, and *Europe Says*. These users may be organizational accounts that output AI information in a long-term and stable manner.

Username	Nick name	Number of posts	Top3 tag
androiddreams	Android Dreams	26596	stablediffusion, aiart, aigenerated
itnewsbot	IT News	3370	ai, biz, machinelearning
remixtures	Miguel Afonso Caetano	2003	ai, generativeai, llms
AlexJimenez	Alex Jimenez	1988	ai, digitaltransformation, generativeai
europesays	Europe Says	1481	ai, artificialintelligence, news
redhotcyber	Redhotcyber	1478	redhotcyber, hacking, ai
byteseu	Bytes Europe	1410	ai, artificialintelligence, news
cryptonewsbot	Crypto News	1364	ai, artificialintelligence, news
alvinashcraft	Alvin Ashcraft 🐿️	1242	ai, dotnet, cloud
dreamplan2501	無夢星球	692	ai, aiartwork, aiart

Figure 2: Top AI content publishers on Mastodon and their most frequent tags.

5.1.4 Graphical result

The line graph in Figure 3 shows the average posting trend of tweets containing the "AI" label in different time periods, and compares the change patterns on weekdays and weekends. As shown in the figure, the number of posts on weekdays is significantly higher than that on weekends, and shows a more regular daytime rhythm. On weekdays, the number of posts on AI topics gradually increases from around 6 a.m., accelerates after 9 a.m., and reaches a peak between 2 p.m. and 4 p.m., with the highest average number of posts approaching 6,500. It then falls slightly in the evening, but remains at a high level until it gradually decreases after 9 p.m.

In contrast, the posting trend on weekends is relatively gentle, and the overall volume is significantly lower. The morning active period is delayed to around 10 a.m. to 12 p.m., and the change range throughout the day is small, lacking a clear peak. This also shows that on weekends, people may get up later, have a slower social rhythm, and have a lower online discussion activity on AI topics.

Overall, the graph clearly reflects the close connection between the time distribution of the topic of "AI" on social media and the rhythm of daily life: the afternoon on weekdays is the main content production period, while activity gradually decreases at night as users rest.

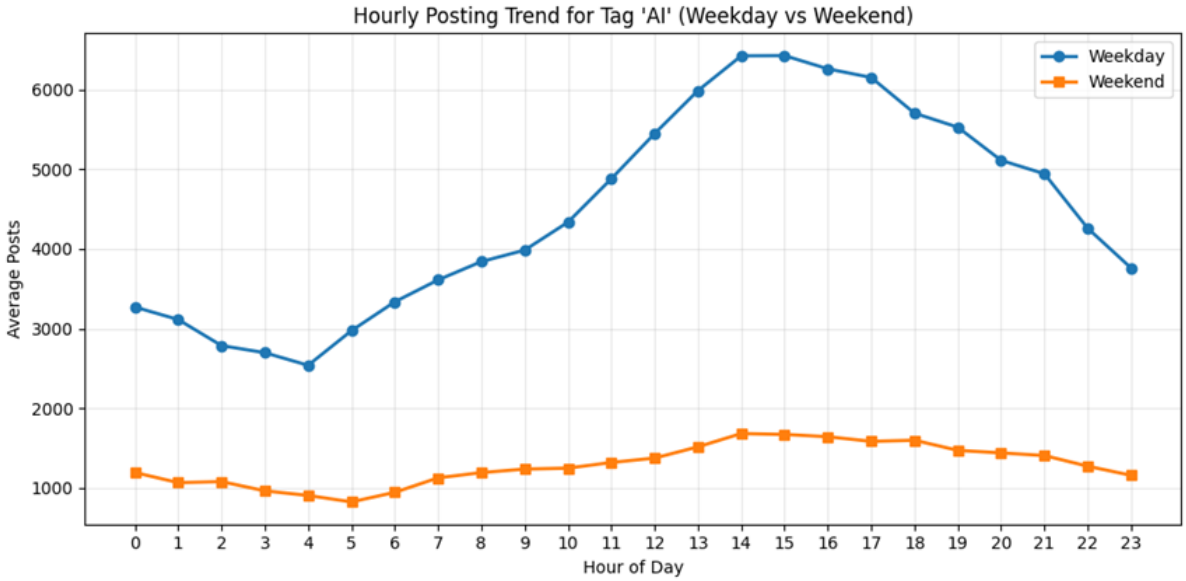

Figure 3: Hourly posting trend of tweets containing the AI tag on weekdays vs weekends.

5.2 Sentiment analysis of AI topics

5.2.1 Scenario description

In this scenario, we conduct sentiment analysis on the content of the post to explore the user's attitude towards AI and related topics (positive/negative/neutral).

5.2.2 Why we choose this

We selected this scenario since emotional changes accurately display users' authentic feelings about AI technology and its events. Monitoring daily average emotional values enables a deeper understanding of how mastodon users' emotions develop over time around AI subjects.

5.2.3 Introduction to VADER modelt

VADER (Valence Aware Dictionary and sEntiment Reasoner) is a dictionary- and rule-based English sentiment analysis tool designed for informal languages such as social media texts (e.g., tweets and comments). It has a large number of built-in sentiment dictionaries, and combined with language features such as capital letters, exclamation marks, negation words, emoticons, and degree adverbs, it can effectively capture the emotional tendencies in short texts.
VADER produces four scores: **positive**, **neutral**, **negative**, and a composite sentiment score called **compound**, which ranges from -1 to $+1$ and indicates the overall emotional polarity of the text.
The classification rules based on the compound score are as follows:

- When compound > 0.05: the text is considered to express **positive** sentiment.
- When compound < -0.05: the text is considered to express **negative** sentiment.

- When compound $\in [-0.05,\ 0.05]$: the text is considered **neutral**.

5.2.4 Graphical result

We used the VADER model to analyze the sentiment polarity of the cleaned Mastodon posts. As shown in the pie chart in Figure 4, in topics related to AI, positive emotions dominate (50.9
This distribution shows that the overall discussion atmosphere of AI on social platforms is positive. When users share new technological breakthroughs (such as the release of GPT-4 and the update of AI painting tools), they often express optimism and excitement. In some periods (such as when ChatGPT is paralyzed and AI causes ethical disputes), there will also be short-term emotional reversals, leading to an increase in negative emotions. Overall, public opinion on AI topics in online communities presents a stable trend with mainly positive sentiment and some local fluctuations.

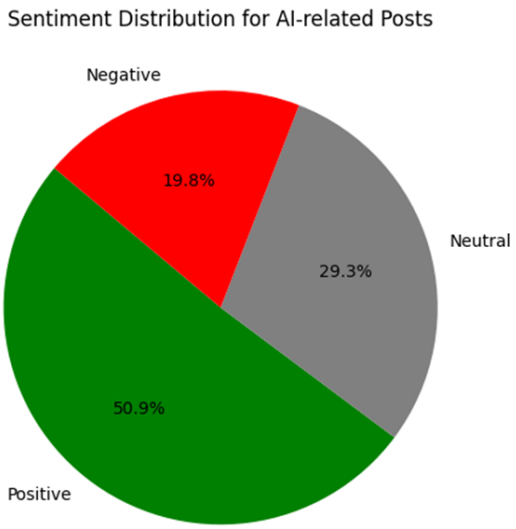

Figure 4: Sentiment distribution for AI-related posts

5.2.5 Extended analysis

To further explore the evolution of social public opinion on AI topics, we have counted the average sentiment polarity (compound score) of daily posts since 2017, and plotted the detailed trends of the full-cycle trend chart.
As shown in the figure 4, the long-term trend shows obvious stage characteristics. From 2018 to 2020, the average sentiment of AI-related discussions was generally positive, and the highest daily average sentiment index was close to +1, showing the public's high expectations and positive cognition of AI technology in the early days. However, after entering 2022, with the accelerated implementation of technology applications and the frequent occurrence of controversial events, sentiment fluctuations have significantly intensified, and have stabilized but significantly declined after 2023. The sentiment center has continued to remain near 0.0, indicating that the overall public opinion attitude has gradually become rational and neutral.

By combining the analysis of major event nodes, multiple emotional high points are highly overlapped with the release of products such as ChatGPT (November 30, 2022) and GPT-4 (March 14, 2023); while the cliff-like decline in emotions occurred around the time when AI painting caused copyright disputes (July 2023) and Google Bard answered incorrectly (October 2023).
Overall, AI-related public opinion has entered the "calm observation" stage from the initial excitement period, and emotions have gradually stabilized but are sensitive, and are easily disturbed by external events (such as policy introduction, system failures, ethical disputes, etc.).

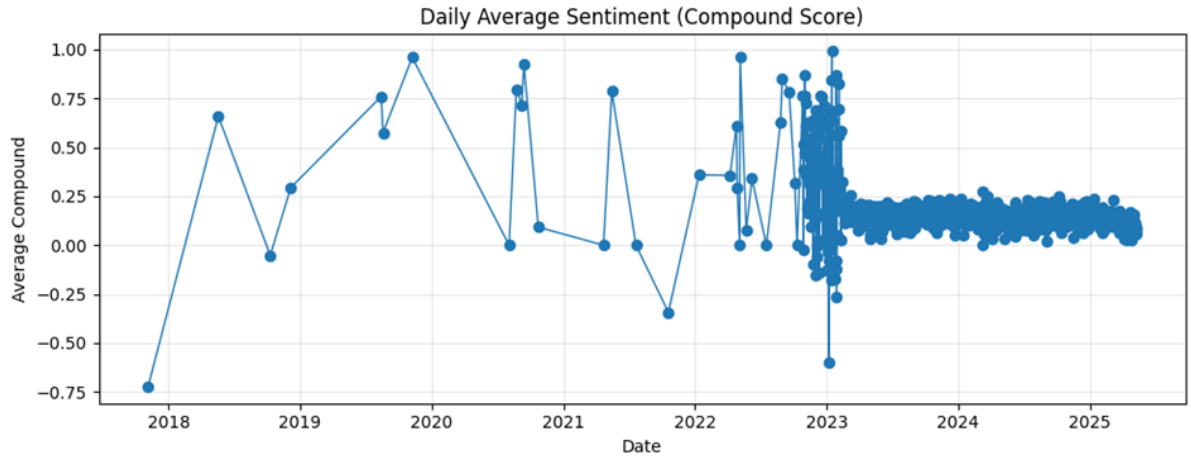

Figure 5: Daily average sentiment (compound score) for AI-related posts

5.3 Heat analysis

5.3.1 Scenario description

In order to deeply understand the changes in the discussion heat of AI topics on social platforms, we conducted a daily trend analysis based on the number of posts containing the "AI" tag on the Mastodon platform in the past year.

5.3.2 Why we choose this

This scenario was selected to examine how AI-related topics fluctuate in popularity on social platforms during specific events. Visualizing the chart enables us to pinpoint the hours when discussion reaches its highest levels.

5.3.3 Graphical result

Figure 6 reveals the dynamic evolution of the daily post volume of the "AI" tag on the Mastodon platform in the past year: the overall trend is a step-by-step increase. It is worth noting that major breakthroughs in AI technology are highly correlated with the fluctuation of post volume: from July to September 2024, a series of major events such as the release of the GPT-4o multimodal API and the technical disclosure of Google Gemini continued to promote public discussion. The discussion volume was relatively stable until

early February 2025. In February 2025, with the official release of the DeepSeek 1.0 ultra-large-scale language model and the exposure of the GPT-5 prototype, the community activity was completely activated, and the daily post volume soared to a full-cycle peak of 430 posts, an increase of more than 200

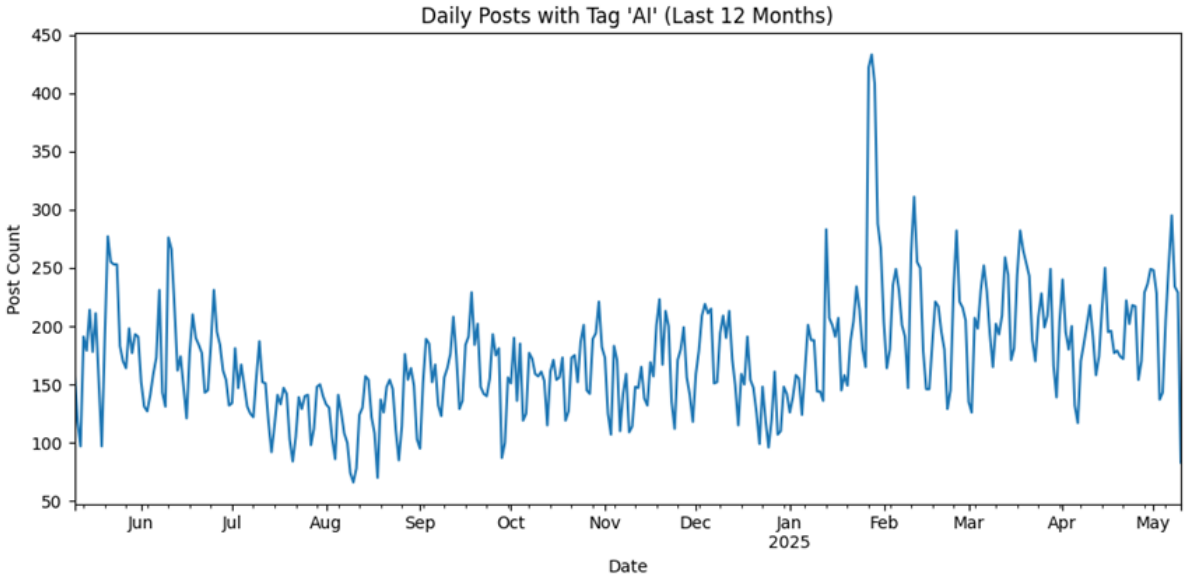

Figure 6: Daily number of posts with AI tag over the past 12 months

5.4 The relevance of the topic of AI to other topics

5.4.1 Scenario description

In order to deeply understand the multi-dimensional interests of platform users in AI, we analyzed posts with the "AI" tag, extracted tags with a high frequency of co-occurrence with AI, and analyzed people's attention to other AI-related fields.

5.4.2 Why we choose this

We chose this scenario to explore the related topics that users discuss when discussing AI, so as to identify the extended focus of AI in different fields (such as artistic creation, automation tools, etc.), and provide support for understanding user interest structure and topic aggregation patterns.

5.4.3 Graphical result

Because the statistics are related topics with ai tags, the ai tag with the highest frequency is excluded.
As shown in the Figure 7 and Figure 8, the statistics of the tags of AI topics on the Mastodon platform show that the discussion heat is highly concentrated on generative technology and creative applications: generativeai, aiart, stablediffusion, aigenerated, generativeart and artificialintelligence occupy the top of the list, indicating that users are most concerned about generative models and their artistic creation capabilities; at the same time, specific product and technology giant tags such as chatgpt, openai, and

google frequently appear, reflecting the continuous tracking of the dynamics of leading manufacturers and new models; and the presence of tags such as privacy, cybersecurity, and machinelearning reveals the community's discussion heat on the security, privacy, and underlying algorithms of AI technology.

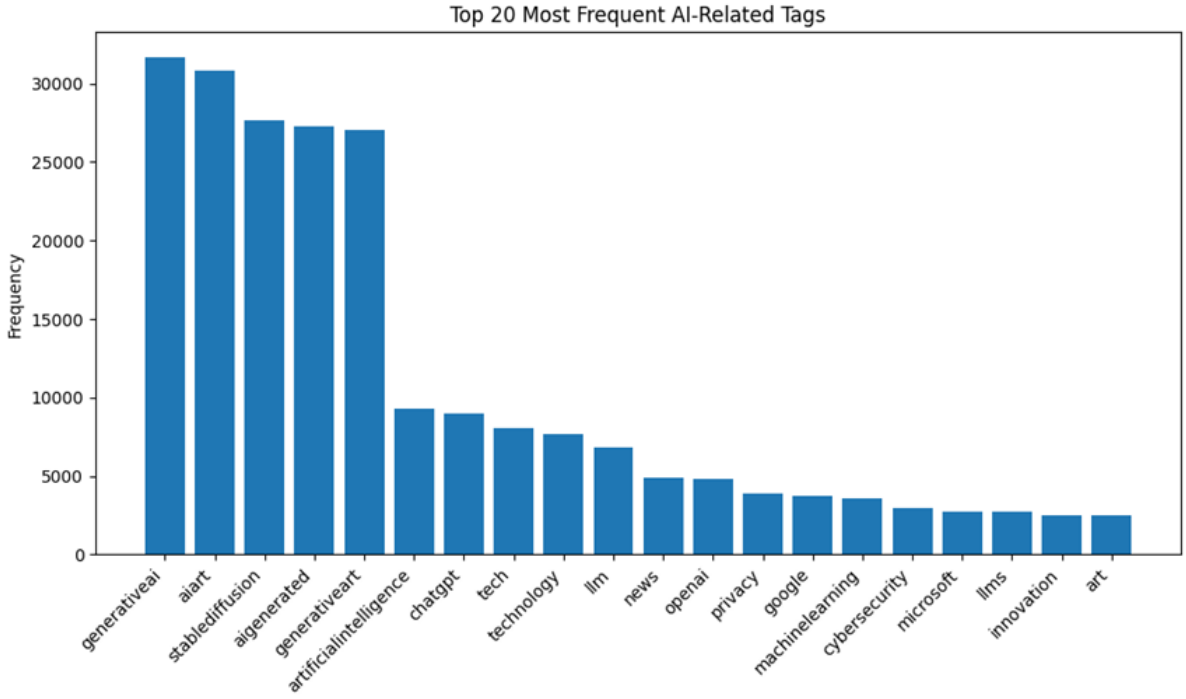

Figure 7: Top 20 most frequent AI-related tags on Mastodon

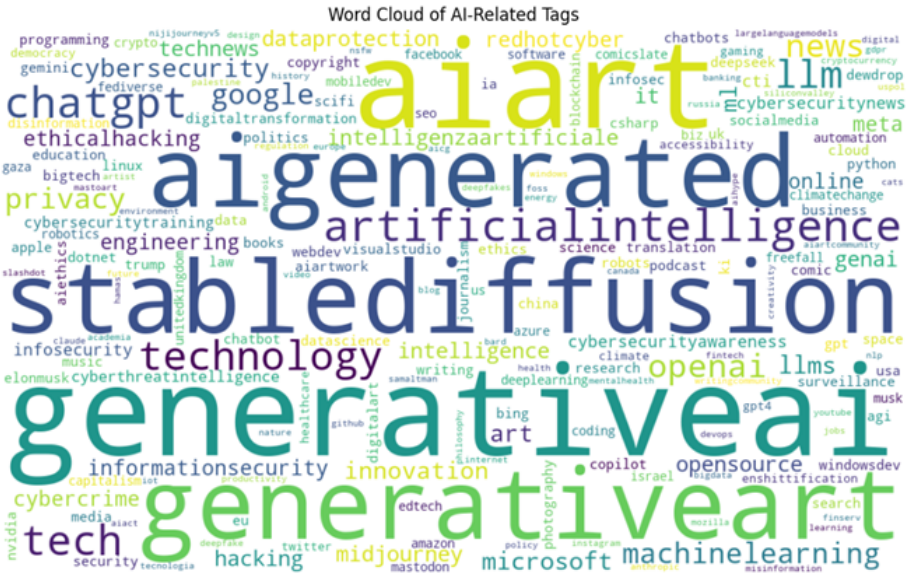

Figure 8: Visualization of co-occurring tags in AI-related Mastodon posts

5.5 Evaluation on NeCTAR Research Cloud & Kubernetes

5.5.1 Pros

1. **Resource Allocation and Scalability**: NeCTAR Research Cloud lets us allocate computing resources flexibly. Kubernetes enables us to scale horizontally by adding more pods when data collection load increases.

2. **Containerization Benefits**: Kubernetes containerization lets us package applications with dependencies, ensuring consistency between development and production and simplifying deployment.

3. **Service Discovery and Load Balancing**: Kubernetes service discovery (accessing Elasticsearch via `elasticsearch-master.elastic.svc.cluster.local:9200`) simplifies inter-service communication. Automatic load balancing distributes traffic across multiple data collector and analyzer instances.

4. **Declarative Configuration**: YAML files for deployment let us define our system architecture as code, making it repeatable and version-controlled. This infrastructure-as-code approach ensures consistent deployment across team members.

5. **High Availability**: Kubernetes provides system resilience through automatic restart of failed containers and cross-node distribution. This is vital for continuous data collection tasks.

5.5.2 Cons

1. **Learning Curve**: Kubernetes has high learning costs. We spent significant time understanding pods, services, deployments, and namespaces before using the platform.

2. **Operational Complexity**: Managing Kubernetes clusters adds operational overhead, and we faced difficulties deploying usable pods. Debugging requires understanding multiple abstraction layers from container logs to pod status to node health, making troubleshooting complex.

3. **Resource Limits**: Despite NeCTAR's valuable resources, we occasionally hit limits when processing large volumes of social media data. Scaling requires careful planning to avoid exceeding allocated quotas.

4. **Network Challenges**: Setting up proper cluster networking, connecting to external social media APIs, and enabling secure application access require complex ingress controller configuration.

5. **State Management Complexity**: Managing stateful applications (like Elasticsearch) in Kubernetes requires additional configuration, adding complexity compared to stateless microservices.

6. **Performance Overhead**: Kubernetes and container virtualization create performance overhead compared to bare metal deployments. Initial frontend memory under-allocation caused slow performance.

In conclusion, despite initial complexity and steep learning curve, the NeCTAR Research Cloud with Kubernetes provides a robust and flexible foundation for our social media mining system. The platform's orchestration, scalability, and resilience suit our dynamic data processing needs.

5.6 Evaluation on Fission

For this project, Fission helped us realize the automation and modularization of stream data collection tasks. Fission's performance in data collection is reflected in the following aspects:

1. Scheduled and automated data collection: The data collector we designed is deployed on k8s by Fission in the form of a package, and a timer is set for each data collector to achieve the function of automatically and regularly pulling data.

2. Functions in Fission do not consume resources as they are not called: The data collector deployed in the cloud only occupies memory when it is called by the Fission prompts. After it is called and finished, the resources will be released immediately, reducing the memory pressure in the cloud.

3. Support for parallel processing of functions: Fission allows multiple data collectors to be called in parallel through timers, greatly improving the collection efficiency.

4. Clear function build and deployment process: Through `requirements.txt` and `build.sh`, developers can quickly specify dependencies and upload functions.

However, Fission also has some limitations:

1. The function on Fission is stateless, which is an advantage, but also a disadvantage, because it cannot keep global variables, such as last_id, so we store the variables in a dedicated Elasticsearch index.Each time the data collector is called, it is necessary to obtain last_id from the Elasticsearch index before continuing to fetch posts.

2. There is a delay in the startup of the function on Fission. The system needs to open the container image and initialize the environment, which is not very friendly to functions that need to return results in real time.

5.7 Evaluation on Elastic Search

In our AI-focused social media analytics platform, we use Elasticsearch as the data storage and search engine for cataloging and searching post collected from Mastodon and Reddit. The following part discusses its pros and cons from our experiences of using it.

5.7.1 Pros

1. **Advanced Full-Text Search and Aggregations.** Elastic Search allows advanced text-based queries, including phrase matching, wildcard, fuzzy by using the Lucene's inverted index, which we use to narrow down AI-related discussions.
2. **Flexible Index Mappings and Custom Analyzers.** By defining mandatory field mappings (e.g., tags, timestamps, content) and by using custom analyzers (HTML stripping, lowercasing, stop-word removal)., we can guarantee the accuracy of our search and aggregation work to implement both topic extraction and sentiment analysis.
3. **Rich Ecosystem Integration.** The native integration of the Elastic Stack with Kibana, and Beats make dashboarding and monitoring easy.
4. **Schema Flexibility.** The design of Elasticsearch indexes allows us to store documents with different structures in the same index, which proved valuable when working with varied data sources like Mastodon and Reddit.
5. **RESTful API and JSON.** The RESTful API with JSON format simplified integration with our Python backend and frontend components allow easy data querying and manipulation.

Elasticsearch's advantages are useful for us to manage and use the Mastodon data.

5.7.2 Cons

1. **Storage Overhead.** The default indexing will also give you multiple formats (inverted indice, stored fields, doc values) that increase storage overhead.
2. **Learning Cost**: The Query DSL is powerful but complex, requiring much time to learn for complex analytical queries compared to traditional SQL.
3. **Near-Real-Time Refresh Latency.** Elasticsearch refresh rate (1 s by default) implies that new indexed documents are not searchable immediately. Because for most dashboards this is M.O.S., but possibly for the sub-second, live event sentiment spy it would be necessary to lower the refresh interval at the cost of higer I/O or to make texture buffering on the client-side.
4. **Monitoring and Maintenance Overhead.** To ensuring cluster health demands continuous monitoring of metrics like CPU, disk I/O, and merge activity. The Elastic monitoring stack deployment itself has overhead of its own.

5.7.3 Opportunities for Optimization

- **Shard and Replica Tuning:**Custom primary shard and replica counts for each index to match query patterns and data growth; leverage index rollover to partition data by time.
- **Dynamic Refresh Interval:**Dynamically set the refresh interval to hot-warm up the indexing throughput / search latency trade-off during bulk load and slow it down for low-latency interactive queries.

- **Index Lifecycle Management (ILM):**Use hot-warm-cold tiering to automatically push older data to cheaper storage nodes.
- **Query Caching and Templates:** Let Elasticsearch query caching and search templates for all your repetitive queries to take the load off the CPU.

Elasticsearch's powerful full-text search, distributed aggregation capabilities, and rich ecosystem makes it a good fit for our social-media analytics solution. But to get the most out of it, you need to tune it well in terms of storage, indexing, and cluster setup for you to make sure that you reduce the costs, lower the latencies, and more while keeping the cluster in a steady state.

6 Challenges & Solutions

1. The reddit API has a limit on the number of posts it can fetch: The reddit API only supports retrieving the most recent 1,000 posts and cannot return more posts. The solution is to use Reddit's Pushshift API, but after our attempts, we found that the Pushshift API is very unstable and cannot fetch information at all.
2. When using fission to create package of data collector, 500 Internal Server Error keeps existing, which cause we can not create subsequent functions. The reason is that the build.sh file edited under Windows uses CRLF line breaks by default. When this file is transferred to the Fission builder pod, **#!/bin/sh** ˚ will be recognized by Linux as an invalid interpreter path, resulting in the inability to find the file. So we just need to convert the line breaks of build.sh to Unix/Linux style line breaks (LF).
3. When we were developing this software locally, we needed to connect to Elastic Search from a Windows host, which was a challenge. We connect to Elasticsearch from our WSL2 virtual machine using an SSH key pair, because Windows does not natively support this connection. However, since our team prefers developing on GUI-based Windows systems and want to access servers from the Windows host. We forward the Elasticsearch port through WSL2 to a local port on the Windows machine. This port-forwarding setup allows our Windows host to reach Elasticsearch as if it were running locally. It took us a while to figure out this connection method.
4. When we need to integrate the front-end and back-end programs and display the output results on the front-end interface, we encountered the problem that the returned result format does not match the front-end program design. Although we have documents to constrain, we found in actual development that the initial design cannot efficiently display our results. In order to achieve comprehensive visualization of the results, the front-end program has been adjusted according to different scenarios, which has reduced the reusability of the code to a certain extent.
5. In the early stages of development, we developed the front-end and back-end programs locally and accessed the Elastic Search service deployed in the cloud through port forwarding, and front-end communicate to back-end on localhost. Obviously, the way the back-end program accesses Elastic Search locally and the way the front-end and back-end communicate is different from when deployed on a cluster.[2] It

took us a long time to communicate and access to the server. The communication api has change from `http://localhost:5000/api/run` to `http://backend-service.backend.svc.cluster.local:5000/api/run`, and the HTTP access to Elastic Search has changed from `https://localhost:9200` (We port-forward the port 9200 on cloud to local port 9200) to `https://elasticsearch-master.elastic.svc.cluster.local:9200`. We need to specify where to locate the service. And because our program is deployed through Docker, every time these URLs are modified, the image needs to be rebuilt and uploaded, and the old pod needs to be deleted before the new pod can be installed, which is very cumbersome and time-consuming.

7 Conclusion

This project systematically analyzed the posting data of Australian users related to "AI" on the Mastodon platform, and completed the full-process data analysis and visualization based on the following four scenarios: user portrait construction, sentiment polarity analysis, heat trend tracking, and topic association detection. The entire process completes data collection and processing through the cloud platform NeCTAR, containerized scheduling Kubernetes, event-driven Fission, and search analysis engine ElasticSearch, and finally realizes interactive visualization analysis on the Jupyter Notebook front end.
In user analysis and portrait recognition, we successfully identified active speakers such as @androiddreams, and combined their label usage distribution to characterize their topic area tendencies, providing a basis for subsequent influence dissemination analysis; in terms of sentiment analysis, with the help of the VADER model to process large-scale unstructured text, we found that the overall discussion atmosphere tends to be positive, especially around major technical nodes (such as the release of GPT-4). There are significant fluctuations; the heat trend analysis reveals that the AI topic has gradually evolved from "stage hot discussions" to "platform core topics". In early 2025, the DeepSeek and GPT-5 incidents triggered community heat; and through label co-occurrence analysis, we further extracted the expansion trend of AI topics in dimensions such as generative content, security, and mainstream platform products.
Our system front end implements interactive visualization based on Jupyter Notebook, which reduces the threshold for complex data analysis and allows researchers or the public to intuitively explore the platform's public opinion structure. This platform has good scalability in both technical implementation and research value.

8 Link

8.1 Video Demonstration Link

8.1.1 ElasticSearch data cleaning

For Mastodon `https://youtu.be/EHHoiPhTA-8`
For Reddit `https://youtu.be/lD9s_GIgmEE`

8.1.2 Cloud-based Solution & ElasticSearch for Analysis

`https://youtu.be/3bW_msDO8uU?si=l5mY5OmJplR2-6Em`

8.1.3 stream data harvesting using Fission

`https://youtu.be/AhEPxCF8R2Q`

8.2 Gitlab link

`https://gitlab.unimelb.edu.au/yutazhou/comp90024_team_17`

References

[1] H. Butler, M. Daly, A. Doyle, S. Gillies, T. Schaub, and S. Hagen. The geojson format. RFC 7946, August 2016.

[2] OpenStack Foundation. Openstack project website, 2020. Accessed: 2020-05-23.

[3] The University of Melbourne. The university of melbourne research cloud website, 2020. Accessed: 2020-05-23.