COMP90024 - Team12

Meilun Yao 1076213

Yingyi Luan 1179002

Yuntao Lu 1166487

Jiayi Xu 1165986

Zheyuan Wu 1166034

# COMP90024 Cluster and Cloud Computing

The University of Melbourne

May 22, 2024

## Abstract

The structure of this report follows the chronological progression of the project. It began by searching and processing various sources from social and data platforms, focusing on finding data with high availability. Particular emphasis was placed on the SUDO and Maston scenarios. Next, It conducted a thorough evaluation of the tools utilized in our project, discussing their strengths and weaknesses. It also includes a detailed presentation of the system's main features, alongside an overview of its basic design and architecture. Following this, we provide a user guide designed to help users effectively utilize and benefit from our cloud services.

The report concludes with a summary of potential limitations and future developments within the project. Additionally, it outlines the collaborative efforts of our team during the implementation of the system.

# Table of Content



**Repository link:**
**https://github.com/lollyluan/COMP90024_2024_ASMT2_Group12.git**
**Video link: https://youtu.be/vJLoTzqmN2o**

# 1. Introduction

**Project Overview and Objectives：** In this project, we are aiming to understand the distribution and sentiment trends of these topics on Mastodon and Twitter, and the correlation between the PM2.5 value of a specific area in Victoria and the asthma rate of that specific region. Our goal was to provide data-driven insights that could inform public health policy and strategies by comparing how these issues are discussed across different datasets. We also incorporated data from the Spatial Urban Data Observatory (SUDO) to enhance our analysis and provide a more comprehensive view of the societal impact.

**Implementation and Data Handling:** The development of this project didn't use too large datasets and complex functionality, so for more efficient management and deployment we used a microservices runtime cluster (MRC), where Kubernetes was used for deployment and orchestration and Fission for API development. This setup allows for robust data collection and streamlined processing. We store all collected data in Elasticsearch to ensure efficient retrieval and scalability for a responsive analytics environment.

**Data Analysis and Front-End Interaction:** The results of our research are presented through the interactive Jupyter Notebook interface, which is the front end of our project, using the python language, which has a very wide range of feature packages that make it easier to present a variety of data effects. This setup allows researchers and the public to engage directly with the data, providing visualizations that describe the trends and results of our analysis. Users can manipulate the data through queries and filters (e.g., by region, time, or specific discussion topics), thus enhancing the exploratory nature of our study.

**Conclusions and Implications:** The preliminary analysis indicated that discussions around news and social issues related to topics' sentiment scores are prevalent on both platforms and often carry a negative sentiment. This reflects widespread public concern over these environmental issues. Our project corroborates the hypothesis that social media discussions can significantly mirror public sentiments and concerns, providing valuable insights into how environmental issues are perceived by the community.

**Project Purpose:** These insights are crucial for policymakers and public health officials as they develop strategies to address environmental health challenges. By understanding the public discourse surrounding air quality and asthma, our project contributes to more effective public health responses and policy formulations. Moving forward, the integration of real-time social media data into environmental health studies could revolutionize how public health impacts are predicted and managed.

# 2. Scenarios Overview

Air quality and the weather is a significant concern for many residents, as it directly impacts public's health, particularly in relation to respiratory conditions like asthma. Therefore, our research involves analyzing the relationship between weather base station data over time, population density, and data with air quality and asthma.

In this project, we focused on analyzing discussions on the social media platforms Twitter and Mastodon, specifically targeting topics related to weather and air quality. Our goal was to compare the distribution of topics and the sentiment trends among discussions across these platforms. By monitoring and analyzing real-time posts and tweets from various regions, we identified key societal concerns impacting public health, especially issues like air quality and asthma.

The comprehensive data collection and analysis revealed that news and social issues were the most frequently discussed topics on both Twitter and Mastodon, often accompanied by strong negative sentiments. This indicates significant public concern and dissatisfaction regarding these matters. These findings highlight the impact of social media on public health discussions, providing policymakers and public health officials with crucial insights into public perceptions of environmental and social issues. This assists in crafting more focused and effective public health strategies. Thus, the project not only enriches our understanding of the health implications of environmental changes but also improves our capability to explore and analyze real-time data through social media.

### 2.1 Scenario analyze

In addition to this, We want to analyze the relationship between meteorological observation base stations and the population of the area. Theoretically, weather stations are for human beings, and the more densely populated the place, there should be more weather stations to obtain more accurate observations. In order to ensure the accuracy and rationality of the data, we selected the weather stations that were recorded in operation during the two-year period from 2021 to 2022, and used their information for statistical purposes. We connect according to the area where they are located, count the number of weather stations in the area, and then analyze them.

For weather stations in an area, in addition to looking at how connected they are by population, they can also be linked to the region's Twitter discussion data. For the weather, the sentiment analysis discussed on Twitter can be used to infer whether the data of base stations in the current area is accurate and whether the number of base stations affects people's evaluation of the weather.

Through this comprehensive analysis, the project expects to provide a new perspective on understanding the specific impacts of environmental change on public health, particularly the link between air quality and asthma. It aims to understand Australian perspectives on air quality and weather. We create data-driven conclusions by analyzing the volume of tweets from specific regions and incorporating data from Fact Monitor Maston. And it is documented and presented through Jupyter Notebook, facilitating a dynamic and interactive display of data and conclusions. This approach not only enhances the accessibility of the findings but also allows for real-time data exploration and analysis.

## 3. Data Collection

### 3.1 Mastodon toots

For mastodon toots data, the project mainly uses Mastodon.au server to collect real-time toots. To receive toots from mastodon.au, the first step is to create an account and then switch to the edit profile section, click the development button on the sidebar and then click the new

application to create a new application. After that, an access token will be automatically generated, which is used to authenticate users to harvest the mastodon toots.

## 3.2 Twitter

Twitter data is all retrieved from the 120GB file that is available from SPARTAN. To accomplish the scenario, for each tweet, sentiment score and topics are also required to be generated. For tweet data in SPARTAN, sentiment score has already been generated, so only topics need to be generated.

## 3.3 SUDO

We used **regional demographic data** from `ABS - Regional Population (SA2) 2001-2020`, and the area code information was used to correlate with other information. The demographic data of this area is very standardized, and there are no missing values, which is more suitable for real data statistics and analysis. This data is sourced from SUDO.
Also, we use Asthma/respiratory data that is downloaded from SUDO, which is called LGA11 Chronic Disease -Modeled Estimate 2011-2013. It is renamed as asthma.json in data/asthma.

## 3.4 ABS MAP

The `SA2_2021_AUST_GDA2020.shp`, this data is the mapping information corresponding to the area code information in SUDO data. It is associated with a region code that provides accurate and detailed information about the boundaries of the region for mapping.

## 3.5 BOM

`BOM-Station` data is Weather Station Directory information in the Bureau of Meteorology. This data records the start and end of all recorded base stations and their location coordinates.

# 4. Data Processing

## 4.1 Mastodon data

Since not all the information is required for the scenario, only 3 features including **id, created_at, content** are selected. Besides, because raw content from Mastodon is in HTML format, BeautifulSoup is used to convert HTML content to plain text and also removes all the hyperlinks, which increases accuracy of sentiment score and topic classification. Also, ZoneInfo is used to convert all the toot to Australian time zone to make all the toots have consistent **created_at** feature.

To achieve the aim of the scenario, for each real-time toot from mastodon, sentiment analysis and topic classification are generated from the **content** of each toot. For topic classification, a pre-trained NLP model called cardiffnlp/tweet-topic-21-multi is used to generate one or multiple topics out of 18 topics in total for the content of each toot. For sentiment analysis, SentimentIntensityAnalyzer in NLTK library is used to generate the sentiment score of each toot. The sentiment score uses the compound polarity score which ranges from -1 to 1 because it normalizes all the positive, negative and neutral scores so that it can best reflect the real sentiment score of the content. Thus, the final features for each toot will be **id, created_at, content, sentiment_score, topic.**

## 4.2 Twitter data

The first step is to filter data from 120Gb twitter data. Since geographic coordinates are required in other scenarios, it needs to store tweets with geographic coordinates. The filtering program runs on SPARTAN with mpi4py used to reduce the running time. Because memory size is limited, the filtered json file is still too large and needs to be stored in batches, which means split the large filtered json file into multiple small json files. After that, all the filtered json files can be copied into local machines from SPARTAN and merged as one json file for further processing. Since we do not need all of the information in a twitter tweet, only six features including **id, created_at, sentiment, text, geo, coordinates** are selected for each twitter tweet. Considering the large amount of data, it will take a large amount of time to generate topics from

a pre-trained model for all the data. Thus, data is further filtered. Firstly, only store tweet data in Victoria. Second, get rid of tweets with text that contains emoji because emoji usually are not related to topic classification. Third, only store tweets with text whose length is over 50 because text with small length such as “Thank you!”, “You are right”usually is not related to a particular topic and longer texts usually contain key terms or phrases that are strongly associated with specific topics. After all the filtering process is completed, topics can be classified using the pre-train model and the process is the same as the one with Mastodon toots. The resulting features for each tweet will be **id, created_at, sentiment, text, topics,geo, coordinates**.

**Map data processing**

Because Scenario needs suburb name to display the map, the suburb name needs to be generated and associate each tweet with a specific suburb using spatial data. The process mainly involves using the SA2 Suburb Shape file. It first calculates the geographic center of each tweet, creates geometric points for these centers using `Point` from the shapely.geometry library , and then creates a GeoDataFrame for each tweet, and then performs a spatial join using `.sjoin_nearest` from geopandas to match each tweet to the nearest suburb based on its geographic center. Finally, it saves the suburb names and other required features to a new json file, so the final resulting features for each tweet will be **id, created_at, sentiment, text, topics, geo, coordinates, SA2_NAME21.**

## 4.3 BOM data

The data we downloaded from the web page is in txt format, and we used python to do a preliminary sorting of the data and store the data in json format. Consistent with the map processing approach used in Map Data Processing in 4.2, a **SA2_NAME21 , State** column has been added to BOM Data for linking with other data. The start and end in the data represent the months and months of normal operation of the weather stations, and this time was used to filter out Victorian weather stations with a start before January 2021 and an end after December 2022 to match the date of twitter data as the object of our study.

### 4.4 Asthma/respiratory data

The json file mainly contains asthma/respiratory rate, area_name, area_code. First, we generate the coordinates from the area_name using **geopy** and then use the map data processing in Section 4.2 to generate the suburb name for each area and only store the data whose area is in Victoria.

# 5. Data Storage

## 5.1 ElasticSearch

Our system uses ElasticSearch to store, search and analyze data. The distributed architecture feature allows it to scale horizontally across multiple nodes, which makes its advantages on handling large volumes of data.

### 5.1.1 Key Features of ElasticSearch

**Index**: The index holds a collection of documents which is used for logical partitioning. The actions like search performed on documents that are inside each specific index. Our system has five indices, those independently store different collections of data.

**sudo**: A collection of documents that store the information of population, state and area.
**bom**: A collection of documents that store the details of climate stations.
**twitter**: The collection of tweets.
**mastodon**: A collection of updated toots (data stream being harvested).
**maps**: A collection of geospatial data of Australia.

**Mapping:** Mapping like a schema in relational databases which defines the different types within the index.

**Document:** Documents stored in ElasticSearch in JSON format. For each document, it contains a unique id and for each field in the document needed to be defined the type of it.

**Shard:** The shard is the atomic part of an index, which can be distributed over the cluster if there are more nodes added to the cluster. For all the indexes in our system, we set the number

of shards as 3 which means the documents of  a single index can be held by three different shards.

**Replica:** The replicas represent the copies of data. If the number of replicas is set by 1, the whole index is on each primary node. If the number of replicas is set by two, the index will be allocated on a primary node and has an exactly same copy of the data on a replica shard on a second node.

**Port 9200:** The default HTTP port for ElasticSearch. A client can communicate with ElasticSearch through this port by sending REST requests.

**ElasticSearch RESTful API:** ElasticSearch provides RESTful API that allows clients to interact with the data and cluster. Clients can use Document API, Search API, Indices API and Cluster API to index documents, search and retrieve data. Our system mainly uses Indices, Document and Search API to manipulate data.

These key features allow ElasticSearch to distribute data across multiple nodes. This distribution enables parallel processing of search operations which can improve the performance. The use of shard and replica are crucial for fault tolerance, these two features make sure that a crush of a single node will not bring down the entire cluster, other nodes can take over the responsibility of the failed node. The replica shards provide redundancy by storing copies of data.

### 5.1.2 Store Data with ElasticSearch API

We can access ElasticSearch by using ElasticSearch API with valid credentials (username and password). The templates of index mapping are stored in `elastic.py`. The datasets are converted into json format for batch processing. The datasets all contain hundreds of thousands of documents, sending a single indexing request might lead to network congestion which will raise the risks of losing data and harder to detect failure. ElasticSearch provides a `bulk` API which can send multiple documents in a single request. The `bulk` request process parallel which takes advantage of the distributed architecture.

# 6. System Design and Architecture

## 6.1 System Architecture

**Melbourne Research Cloud:**

Kubernetes relies on cloud infrastructure for compute nodes, disk volumes and network resources. Openstack is used to provision those resources and deploy kubernetes control planes on cluster nodes. In order to access k8s with CLI clients, we should first login the university's VPN, and then create an SSH tunnel with a bastion node to use CLI tools. Lastly, we use k8s port forwarding to access the services and pods.

**Pros and cons of Unimelb research cloud(MRC)**

Pros:

- Since MRC is a private cloud, it has greater control over the activities happening on the cloud and thus it is easier for MRC to identify any security flaws
- Reliable and more trust
- Free to eligible users compared with cloud providers such as AWS

Cons:

- MRC has a finite pool of IP addresses available for allocation. Given the high demand and limited supply, these IP addresses can quickly be exhausted.
- Instances in the MRC do not have public, externally-accessible IP addresses, meaning they cannot be accessed directly from the internet. This is because MRC is designed to be an internal cloud service for the University of Melbourne.
- Access to MRC instances is restricted to the University of Melbourne network. To connect to these instances, users must be within the university's network or use the UNIMELB AnyConnect VPN, which provides secure remote access to the university's internal network.
- In MRC, once a key-pair is associated with an instance, it cannot be changed on the fly. To update the instance with a new key-pair, it is required to relaunch the instance with the new key-pair.
- The University of Melbourne may need to invest an extra amount of money for management, maintenance of hardware of MRC.

**Kubernetes:**

In our project, we use kubernetes, which is a container orchestration platform. It manages resources on multiple computer nodes and we can use yaml files to define the system configuration. We run the private k8s clusters with Melbourne resource cloud, and the resources are provisioned using openstack.

In our project, we use an elasticsearch cluster consisting of 4 nodes(1 master node, 3 worker nodes) as well as another node running bastian. Whenever we created nodes, pods automatically created. Those pods are used to hold containers, container images, environment variables and volumes. ConfigMaps are used to set environment variables for the pods. This can allow us to alter environment variables independently without ruining the code and functions.

**Fault tolerance**

In the elasticsearch, we used 3 worker nodes and 1 master node in order to guarantee fault tolerance. If one node fails, the other nodes can rebalance and work normally. Replicas are also used in order to achieve this goal. We have one or more pods in each node, running different services. A fixed route path(eg. elasticsearch-master.elastic.svc.cluster.local) is attached to those services, routing all external HTTP traffic. If one of the pods fails to run properly, a new pod can be created automatically to replace the original pod without updating the ip address.

**Workflow:**

In our project, we fetch data from external websites first. After a series of data processing(eg. data warehousing, data cleaning), those data is stored in elasticsearch. Main data and replicas are saved in order to guarantee consistency and partition tolerance. We use fission to build functions and routers. The client(jupyter notebook) can therefore access the data through rest api.

**REST API:**

REST API is used to connect frontend with backend. Through requesting a predefined URL, the client can access the result resources from the server.

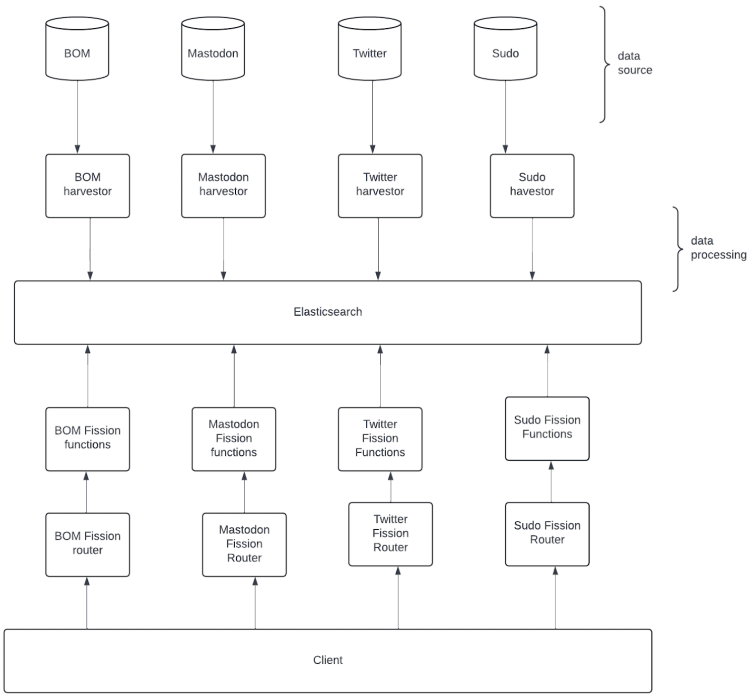


Figure 1: Architecture of whole system

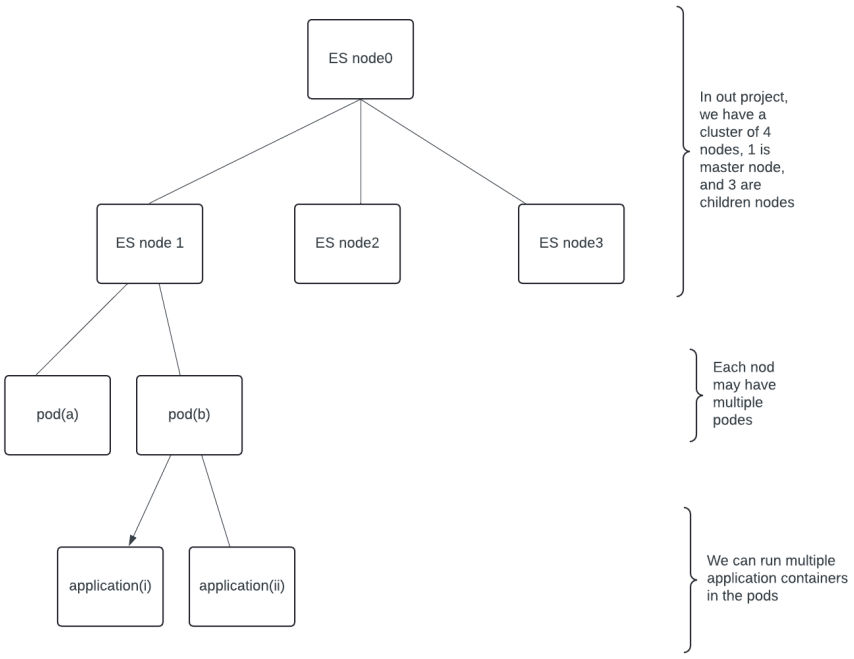

Figure 2: Structure of clusters

## 6.2 Fission

Fission is a serverless framework built on top of Kubernetes that allows developers to quickly create and run microservices and functions within a Kubernetes cluster. This framework enables developers to focus on writing business logic rather than managing infrastructure. The project uses the Python language and offers a series of API endpoints for managing and deploying functions. We deployed

First, by using SSH tunneling and Kubernetes port forwarding `'ssh -i ./<private_key> -L 6443:"192.168.10.12":6443 ubuntu@172.26.128.21'`, we establish a local-to-server connection that allows services running on the Kubernetes cluster to be accessed from the localhost. In order to interact with ElasticSearch, we set up elasticsearch : `'kubectl port-forward service/elasticsearch-master -n elastic 9200:9200'`, `'kubectl port-forward service/kibana-kibana -n elastic 5601:5601'`, then connect to the router: `kubectl port-forward service/router -n fission 9090:80`, then team members can get api requests.

These Fission functions can be directly called by Jupyter Notebook for data visualization purposes. In the Fission functions, we have implemented logic to store processed data directly into Elasticsearch. This involves constructing a JSON document and utilizing Elasticsearch's REST API for data insertion. The settings in the Fission functions include connection parameters and other configurations such as timeout settings and connection pool sizes.

We also use a ConfigMap to store authentication details securely, preventing the exposure of sensitive information in the code. ConfigMap can also help reduce coupling between functions and environment variables; we can update those variables easily without bothering with the source code and functions we built. We create several ConfigMap to store ElasticSearch URL, which can make the document look neater, more readable, with clear sub-functions.

Our team consists of two members collaborating on this part. During the collaboration, we

discovered that deploying Fission only requires one member to set up the environment initially, and then the other member can deploy the necessary functions. This situation can lead to confusion and potential conflicts if team members inadvertently redeploy environments or Fission, thus crash each other's environments. Therefore, it is crucial to have a clear understanding beforehand to avoid such conflicts.

The setup involving Fission within a Kubernetes architecture notably enhances efficiency, flexibility, and scalability in data processing and visualization workflows. It reduces coupling, and its modular approach allows each function to work separately; if one function does not work, the other functions in the system will not be affected. Also Fission only active when triggered and remains idle otherwise, which optimizes resource usage and reduces costs. Therefore it is very stable and it can effectively manage large-scale data workflows

**Fission data ingestion**

We created a function to fetch epa data from external websites https://portal.api.epa.vic.gov.au/ post the average value of PM2.5 per hour and coordinate to elasticsearch. Then we created a timer to execute that function every hour. We can validate using the fission commands. This allows us to fetch and post data automatically. If we want to update the interval time or functions, we can update the timer without modifying the original functions.

# 7. Data Visualization

Scenario 1: Topic comparison between Twitter and Mastodon

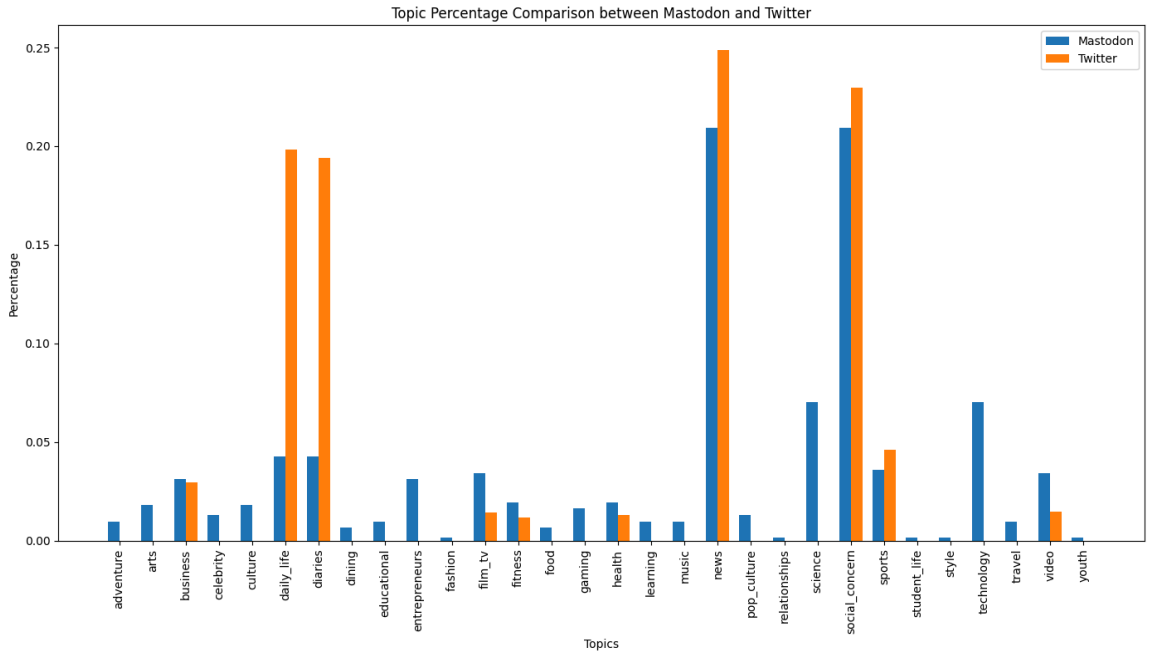


Figure 3: Distribution of various topics in Mastodon and Twitter data

Figure 3 illustrates the topics percentage comparison between toots in Mastodon and tweets in Twitter. Some of the twitter topics are missing because of the filtering as discussed in ??data preprocessing. However, for topics which both Mastodon and twitter have, it is obvious that daily life and diaries account for much higher proportion than the ones in Mastodon, while news and social concerns both account for a high proportion in Mastodon toots and Twitter tweets. The reasons for that are probably because Mastodon toots are stream data and the recent unpeaceful events including wars in the world largely draw people's attention towards these events. Also, it is interesting to note that technology also accounts for the third highest proportion which align with the theme of recent events. The reasons for the high proportion of topics including diaries and daily life in twitter tweets may be due to the nature of Twitter, which is a social networking site.

Scenario 2: Sentiment Comparison of topic in Mastodon and Twitter

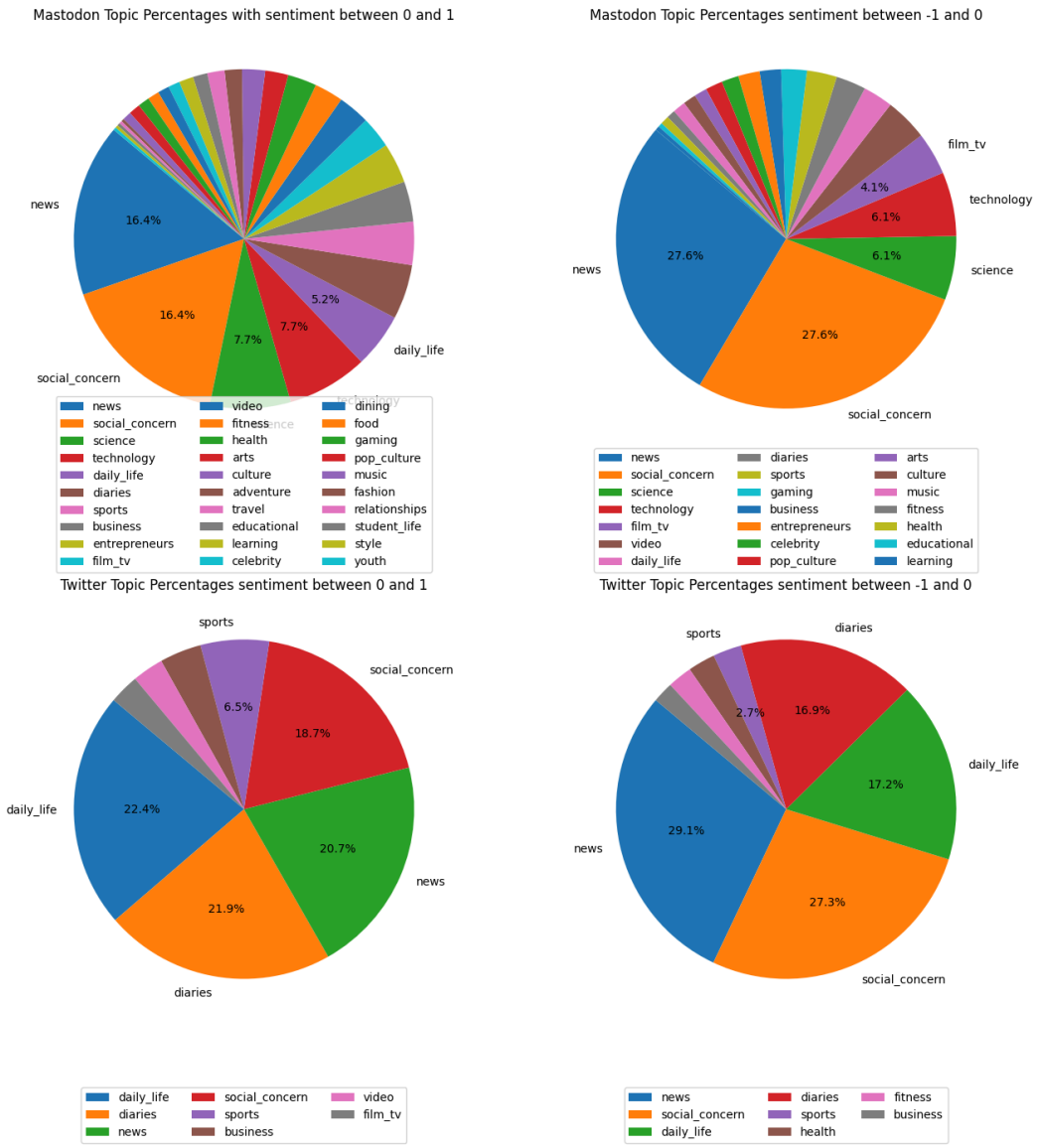


Figure 4: topic percentage in Mastodon and Twitter data with sentiment range 0-1 and -1-0

Figure 4 illustrates the topic distribution with positive sentiment and negative sentiment in Mastodon and Twitter respectively. Regarding Mastodon, it is clear that topics including news and social concern have more negative sentiment than the positive sentiment, which aligns with findings of figure 3 because of the recent unpeaceful events that cause more negative feelings than positive feelings to the people. Besides, it is interesting to note that tweets in Twitter have the same trend that topics including news and social concern have more negative sentiment, while topics including daily life and diaries have more positive sentiment in both Mastodon and

Twitter. It may be because topics such as news and social concerns evoke more negative sentiment due to their serious and conflict-driven nature, while daily life and diary topics have more positive sentiment because they often involve joyful moments and relaxing discussions.


Figure 5: Map with weather station and region split in Victoria.

The red clouds on this map are all the base stations in operation between 2021 and 2022, and they are scattered in various areas of Victoria, and it can be clearly seen that some of the base stations in the middle are densely distributed and the surrounding areas are gradually thinning. The Northwest Territories border other states on a piece of land with few base stations, and the weather information in this area is predicted by the surrounding weather stations. A similar situation occurs in the eastern coastal areas. The areas that are clearly masked below are SA2-based areas, with the boundaries of each area separated by a blue line.

As Figure 5, If you zoom in on the map, clicking on some of the regions will give you a popup like the one shown in the image, which contains the name of the region, the population density per square kilometer, the number of stations, the number of Twitter data that the region has, and its average score. This information is linked based on the current map data for the area. Since

we are using a map of the SA2 zone division, which results in some areas being small in size, we use population density to represent the population information of the current area, rather than using the total population.

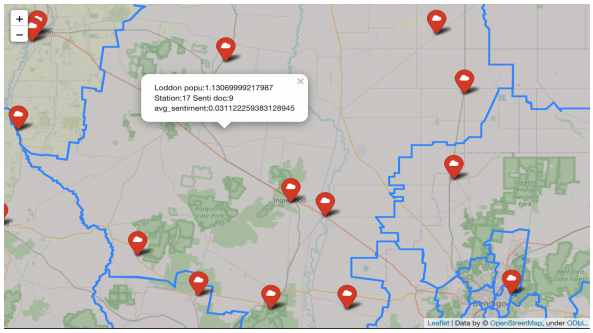


Figure 6: Region popup in Map.

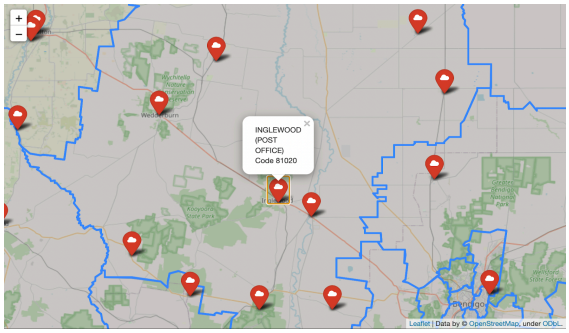


Figure 7: Weather station popup in Map.

And as shown in the figure 7, click on the location of the weather station, and the pop-up window of the weather station contains information such as the name of the station and its number in the Bureau of Meteorology.

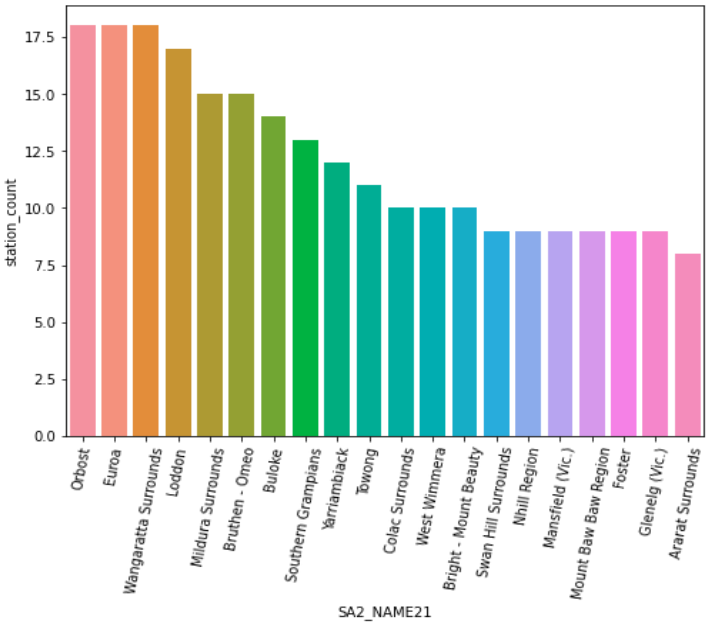


Figure 9: Rank in station count (top20).

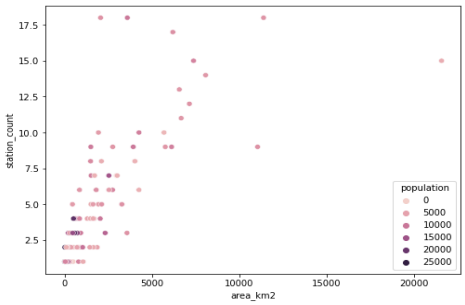


Figure 8: Station v.s. Population

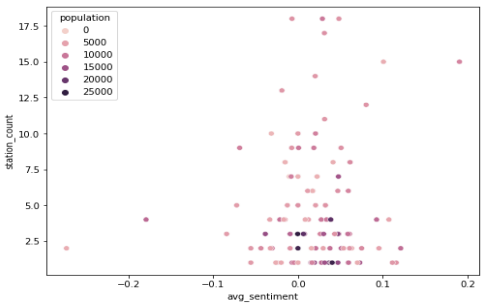


Figure 10: Station v.s. Twitter Sentiment.

Scenario 3: If Population or Popular density infarct Weather Station number in one region?
Figure 9 shows the 20 regions with the most weather stations in the region, with Orbost and Euroa having the most weather stations at 18. We speculate that more people should have more weather base stations in places where more people live, but in this image, we don't see any familiar areas. To further verify this idea, we add other variables, such as the size of the region and the total population, as shown in Figure 8.

There is no clear correlation between the population information and the weather stations in Figure 8, and it can be seen that the colors in the graph do not show a trend, but are scattered, and there are some more populous cities scattered in areas with fewer base stations. In other words, to a certain extent, the large population will affect the data of the meteorological base stations in the area, which has not increased as previously speculated, but decreased. Rather, it is the area of the area that is more relevant to the weather base station, and the wider the land, the more weather stations. Since the regional division we use is SA2, it is divided according to population density to some extent, and there are more dense divisions in relatively densely populated areas. That's why it appears in Figure 6 that there are no well-known densely populated areas in the top 20 regions.

Scenario 4: Correlation of  Weather Station and Twitter Sentiment in one region
Image 10 As we speculated, the larger the number of weather stations, the closer the Twitter discussion data is to a neutral positive evaluation. But since we don't have access to Twitter weather data, this data doesn't support our initial view. Just known places with fewer weather stations, some of the populations are large. What we can get from this is that in those areas with large populations, their average sentiment is always neutral and directly floating. This is due to the large amount of data that has been de-dried for the region. So overall, the data on Twitter is relatively neutral to a certain extent.

In addition to this, our group would like to emphasize one design: our API is designed to be interactive. In the Jupyter Notebook presentation, our group chose the Folium Package to draw the map. We didn't find a better way to call the API at the moment of interaction, so in this jupyter notebook showcase, loading data upfront is not as fast, and we have to run through all

the regions in turn at the beginning to get the complete data. But if it's an interactive front-end system, this design will reduce the size of the front-end cache data, and the loading data and access speed should be within a comfortable range.

Scenario 5: relationship between PM2.5 and asthma/respiratory disease rate

First of all, get the data from asthma and epa api. After the inner combination, the data only left four regions. It was caused by incomplete data, though we got 47 and 24 respectively on asthma and epa data, only four inner here. Because the epa api is real-time, so do aggregation group data for those four regions by suburb name, and take the average value of it to analyze.

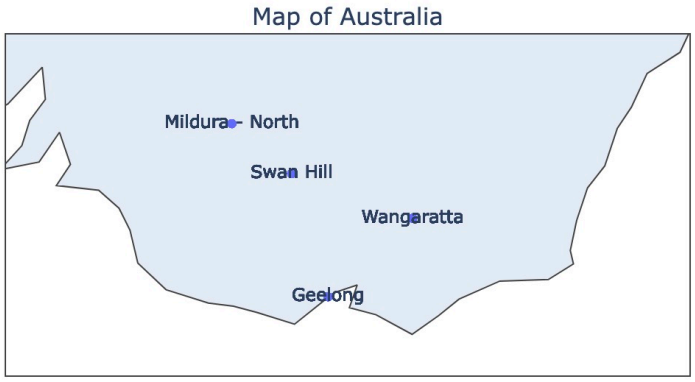


Figure 10: suburb has both asthma rate and PM2.5 average value

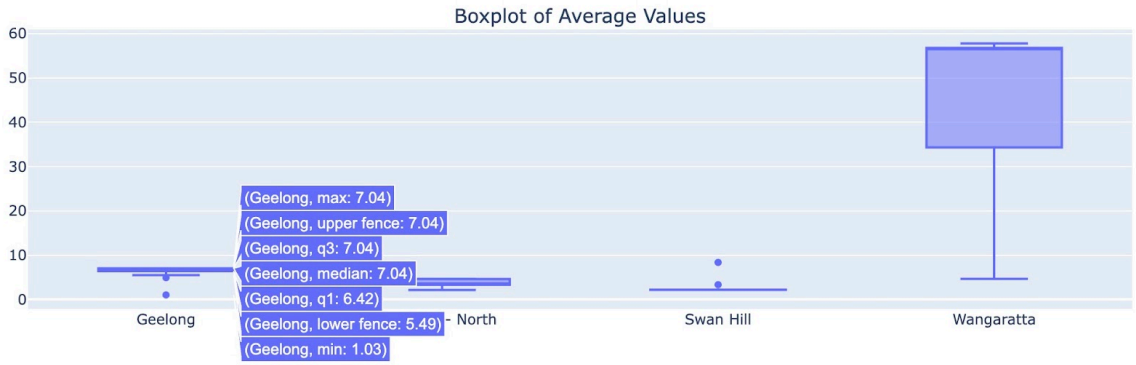


Figure 11: boxplot of average value of Pm2.5 in different areas(Geelong)

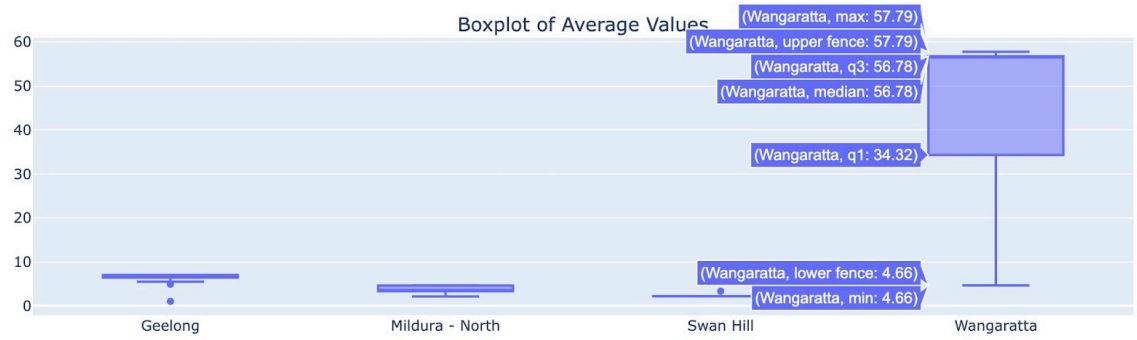


Figure 12: boxplot of average value of Pm2.5 in different areas(Wangaratta)

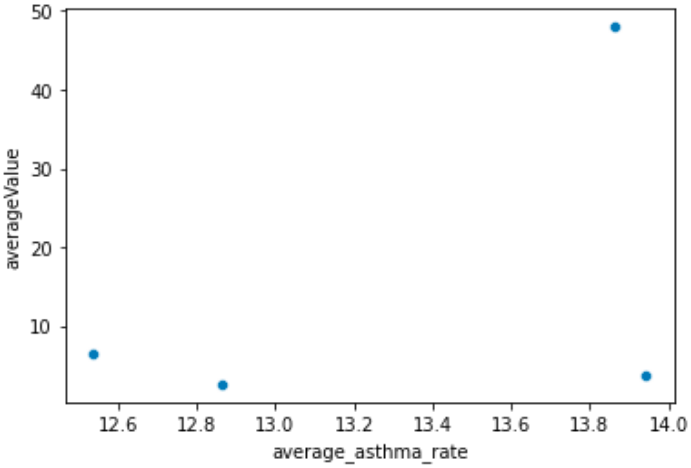


Figure 13: average value of Pm2.5 vs average asthma rate

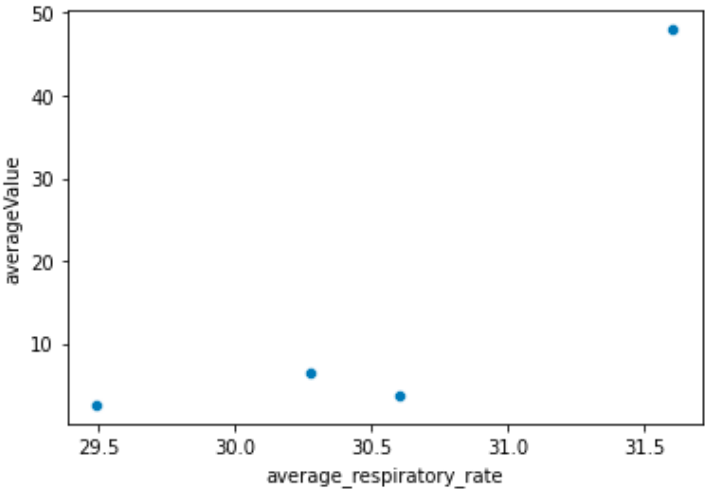


Figure 14: average value of PM2.5 vs average respiratory disease rate

Figure 11 and 12 shows the boxplot of average value of Pm2.5 in different areas, it is interesting to see that Wangaratta is much higher than the other three areas, because the values are recorded from only a small period of one day , it may be due to sudden whether change or events such as explosion in factories that cause the spike in PM2.5.
From figure 13 and figure 14, because there are not many areas who have both asthma/respiratory rate, it is hard to detect a trend. However, in figure 14, there seems to be a trend that the higher the average value of PM2.5, the higher the average respiratory disease rate. This trend is not obvious in Figure 13, but it is likely that if there are enough regions, the higher the average value of PM2.5, the higher the average respiratory rate, the higher the average asthma rate.

# 8. Testing

Our team created an end-to-end test script using Python's unittest framework. This is primarily used to test various features of the API service. The code consists of two main parts: the HTTPSession class, which is used to manage HTTP sessions, and the TestEnd2End class, which is used to define and execute specific test cases.

## 8.1 Class HTTPSession

This class is a simple wrapper for an HTTP client used to send HTTP requests. It has the following functions:
__init__: Constructor that initializes a requests.Session() and sets the base URL, which in this case consists of the protocol, hostname, and port number. get, post, put, delete: These methods correspond to HTTP GET, POST, PUT, and DELETE requests, respectively. Each method accepts a path parameter and optionally data, which are used to send the request to the specified URL.

### 8.2 Class TestEnd2End

This class inherits from unittest.TestCase and is used to define and execute a series of test cases. Specific test cases include:

**Tests for Mastodon and Twitter:** These tests check whether GET requests sent through different paths (/mastodon/gt, /twitter/lt, etc.) return a status code 200 (i.e., a successful HTTP response), whether the returned content is non-empty, and whether the length of the returned JSON object is greater than 0, which indicates that the response contains data.

**Tests for specific queries:** e.g. testing Twitter's average sentiment score (/twitter/sentiment) or a specific count (/twitter/count) to ensure that the response contains the correct data.

**Tests for other functionality:** e.g. testing a list of weather observatories (/get-bom-list), map region information (/get-map-region-info) and population lists (/get-population-list). These tests check that the information was successfully fetched and that it is in the correct format. In the final part of the script, an HTTPSession instance is first created to send requests to port 9090 on the local host. The execution of all test cases is then initiated by calling unittest.main().

This test method is mainly used to verify that the API works as expected and returns the correct data and HTTP status codes, is a common automated test used in development to help ensure the stability and reliability of an application during development and deployment.

# 9. Team Collaboration

Our team uses Zoom for weekly meetings and GitHub to manage task deliveries. Team members are assigned to different modules based on their professional backgrounds and skills, ensuring efficient project progression and quality.

**Data Collection and Scenario Analysis :**

- **Zheyuan Wu:** Responsible for collecting data from Mastodon and Twitter, finding correlations between them. He processes this data and visualizes it using Jupyter Notebook.

- **Jiayi Xu:** Searches for data related to weather and weather stations on the SUDO platform and displays their geographic locations using the mapping functions in Jupyter Notebook.

**Data Storage & Testing:**

- **Yingyi Luan:** Handles data storage using Elasticsearch, including database mapping and querying the database through Fission, also testing all the functionalities.

**Cloud Infrastructure and Function API:**

- **Yuntao Lu:** Gains a deep understanding of cloud platform architecture, sets up the Kubernetes environment, and writes the necessary functions.
- **Meilun Yao:** Coordinates team members in using Fission collaboratively and completes the function APIs for the project scenarios.

This division of workload ensures that each member is optimized for their respective area of expertise, and that the project is efficiently synchronized throughout the entire process, from data collection, processing, storage to presentation. Team members can learn what they need to learn at the same time, and then explain what they have learned to the others during meetings. Through this collaboration, our project can systematically process and analyze data to support decision-making and strategy development.

# 10. Challenges and Limitations

**Jupyter notebook**

First, it is hard to determine the location of the toot in Mastodon server because Mastodon does not provide location-related data, so the scenario our team can think of is to compare the sentiment and topic generated from the content of toots in Mastodon and tweets in Twitter. Thus, for a location-related scenario, our team decided to use data from Twitter instead of Mastodon. Second, due to the time constraint, it is hard to generate the topic for all the tweets in Twitter because of the large amount of tweets(100GB) of twitter and the time required for the pre-trained model to generate the topic, so our team decided to only display data in Victoria instead of Australia.

**Fission**

In the project, we faced a lot of challenges and limitations by using fission. Even though it has good modular approach, it is so complex in monitoring and debugging. Firstly, It needs to be updated every time a change is made, and it's also hard to track the path of the request through the system and understand what went wrong. For example, in the project, we want to compare the features of the two datasets to filter the corresponding on the data, and then query information, so we want to create more than one service in a python file, but found that it is very difficult to operate, once there is a problem, it is not at all clear that where it is a place, so we need to do the service decoupling, turned each service into individual test, which is very troublesome. It takes a lot of effort to find out the exact cause of the problem. This is especially problematic to debug when it comes to functions that need to get real-time updates.

In addition, Fission has limited control over the environment. We use a lot of python packages when doing data processing, and the data is handled well, but when starting to deploy these functions with Fission, we spend a lot of time on version control, we initially wanted to fetch real-time data from Mastodon, but the Torch package was used to process Mastodon. However, the team members responsible for this part of the project did not have the means to run the specific version, the dependency management is a big challenge.

**ElasticSearch**

ElasticSearch has advantages in handling large volumes of data and adding nodes to the cluster can improve the storage capacity. This shows the extensibility and scalability of ElasticSearch. ElasticSearch can handle geospatial data like Polygons which can be used to search the region of a specific point provided. The RESTful API provides the convenience of indexing and searching, however, with large index fields like map data, the querying time will increase rapidly and require more complicated search queries. Moreover, Elasticsearch needs effort on maintenance, which includes index management, shard allocation to ensure the health conditions of the cluster. Another challenge of using ElasticSearch is, sometimes it is hard to update the index mapping. Adding or removing a data field is easy to implement with existing API, but if we need to change the data type from one to another such as from "text" to

"keyword", which will lead to an error with returning a bad request. Therefore, we need to delete and re-upload the data again, which takes time if with a large dataset.

The limitation of ElasticSearch could be the consistency of data. We found when one team member is uploading documents to an existing index, other team members could still query the data simultaneously. In this situation, it will lead to the inconsistency of data while different numbers of documents have been queried.

**Git**

```
git clone
git branch <branch-name>
git status
git commit -m "describe the changes"
git push origin <branch-name>
git pull origin <branch-name>
git merge <branch-name>
```

Our team uses GitHub for version control which allows us to track changes, branches and merging. To create and deploy fission routers, fission uses independent function files which makes it easier during development, which means there was less conflict for editing the same file. However, there are still some limitations while we try to upload large files. Switching branches in GitHub can lead to potential loss of changes if someone is not familiar with GitHub.