Big Data Analytics on the Cloud

What do Australians think of Donald Trump and the use of tariffs in Mastodon?

Team Name: COMP90024 Team35

Zijing Li (1534562)
Hengjin Hou (1514311)
Binzhen Wei (1575618)
Wenquan Wan (1503051)
Zeyu Chen (1266160)

Gitlab Repository: https://gitlab.unimelb.edu.au/HENGJIN/comp90024-2025-35
YouTube Video: https://www.youtube.com/watch?v=8p3OrnrJdpY

THE UNIVERSITY OF MELBOURNE
May 2025

Introduction

This project aims to develop a scalable cloud-based big data analytics system that can collect, store and analyze social media content in real time using some cloud computing technology. The system relies on the NeCTAR Research Cloud Platform and combines tools such as Kubernetes, Fission and ElasticSearch to explore related patterns on major political and economic issues related to Australia.

Scenario

Our team's analytic scenario is **"What do Australians think of Donald Trump and the use of tariffs on Mastodon?"**. We chose this topic because it is closely related to Australian politics and economics, especially for the Australian - American global trade. This project relies primarily on the Mastodon platform data. Its decentralized structure gives us access to a large amount of user-generated content, reflecting a diverse range of viewpoints relevant to Australia's position.

Background

Donald Trump's series of trade tariff policies during and after his presidency have broken the traditional international trade pattern. As a country that is highly dependent on exports and diplomatic relations, Australia is particularly sensitive to such global changes written in [3]. Therefore, understanding the public's reaction to these events will help to deeply assess the social cognition and political attitudes of the Australian people towards international affairs. In addition, as a decentralized emerging social platform, the emotions expressed by Mastodon users provide us with a unique perspective, which is different from the public opinion ecology of mainstream platforms such as Twitter or Reddit, thereby revealing a more diverse and decentralized public discourse.

Relevance

This scenario is closely related to the real world in several ways. First, tracking public sentiment is of great significance for government governance and policy research. Government departments, media analysts, and policy researchers often need to understand the public's views and attitudes on foreign leaders and their policies in order to make more targeted decisions and judgments. Secondly, understanding the public's perception of trade and tariff policies is also important. Understanding the Australian society's response to relevant policies will help to formulate a more forward-looking trade diplomacy strategy and optimize the government's expression in public communication in [1]. Finally, focusing on specific social platforms can provide unique insights. This project chose to analyze the decentralized social media platform Mastodon, hoping to use it to explore niche voices and grassroots public opinion that mainstream media may ignore, thereby enriching our overall understanding of social public opinion.

Through this analysis, our goal is to show how Mastodon data can be used to gain insights into international political perceptions, and in particular, how Mastodon can play a unique role in shaping this narrative.

System Architecture and Design

This section outlines the architecture of our cloud-based big data analytics system. This architecture was designed with modularity, scalability, and reliability, mainly using Kubernetes as the orchestration platform and Fission as the function layer. The system consists of multiple coupled components that handle data collecting, querying, storage, and visualization. Figure 1 shows the overall architecture we developed for our system, including service interactions and data flow across major components such as Fission functions, ElasticSearch, and the Jupyter Notebook.

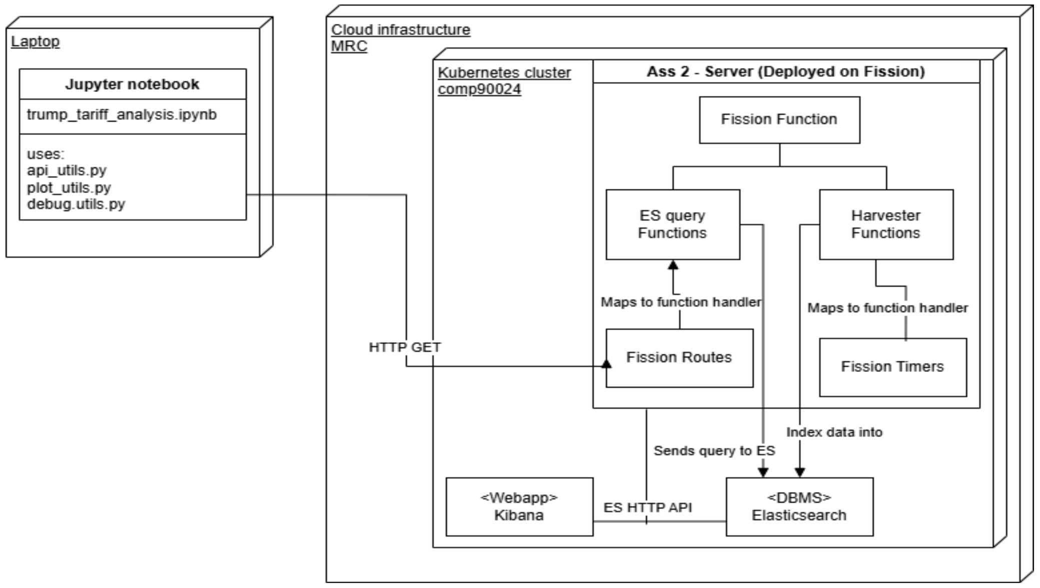
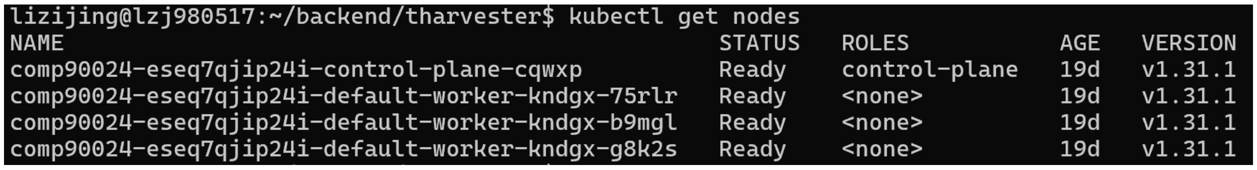


Figure 1. The System Server diagram (UML)

Overview of Cluster and Service Deployment

Our system was deployed on the NeCTAR Research Cloud using the Kubernetes cluster consisting of one master node and three worker nodes, which was deployed by the “openstack coe cluster create” command to provision a cluster named “comp90024”. And each node was configured using the “uom.mse.2c9g” flavor (2 CPUs, 9GB RAM) as shown in Figure 2, which ensures sufficient resources for our real-time data harvesting and other tasks.

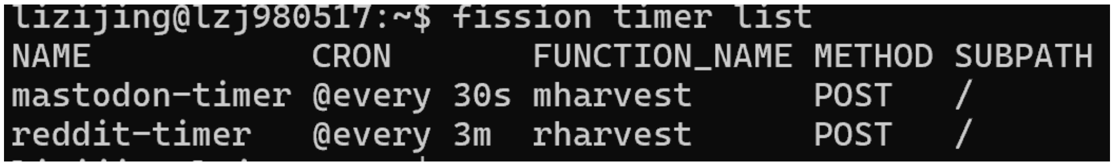


```
lizijing@lzj980517:~/backend/tharvester$ kubectl get nodes
NAME                                          STATUS   ROLES           AGE   VERSION
comp90024-eseq7qjip24i-control-plane-cqwxp    Ready    control-plane   19d   v1.31.1
comp90024-eseq7qjip24i-default-worker-kndgx-75rlr   Ready    <none>          19d   v1.31.1
comp90024-eseq7qjip24i-default-worker-kndgx-b9mgl   Ready    <none>          19d   v1.31.1
comp90024-eseq7qjip24i-default-worker-kndgx-g8k2s   Ready    <none>          19d   v1.31.1
```

Figure 2. Kubernetes Cluster Nodes

All internal services were placed across the cluster nodes using Kubernetes' built-in scheduler based on resource availability and node health. This distribution can be checked by using the “kubectl get pods -A -o wide” command. The scheduler made this distribution to ensure load is spread across all available worker nodes and resource-intensive services are isolated. To prevent external exposure, the system was constructed as a private cluster model without public IP addresses. All external access was routed through a bastion host. For example, Elasticsearch and Kibana can be accessed using “kubectl port-forward” command from the bastion host. We deploy Elasticsearch and Kibana via Helm charts into the elastic namespace using the 8.5.1 es-version, which simplified installation and allowed configuration of resources such as replica count through command parameters. Helm allowed us to easily configure parameters such as the number of Elasticsearch replicas and persistent volume size. Overall, our system architecture provides strong network isolation, with no public exposure of internal APIs or dashboards.

Within the system, Fission functions serve as a compute layer, interacting with Elasticsearch in real time. As shown in Figure 1, Fission's HTTP routes connect the different Fission ES query functions such as "mstats" and "rstats", which are both deployed by YAML specifications. These Fission functions parse parameters from HTTP headers and do different kinds of Elasticsearch queries for data in "mastodon-posts" and "reddit-posts" indexes, returning results as a JSON array. In addition, our system also includes multiple data harvester functions deployed by Fission YAML specifications. These harvesters, such as "mharvester" for Mastodon and "rharvester" for Reddit, are triggered periodically by using Fission timers as shown in Figure 3. Once triggered, each harvester queries its respective platform's API, processes needed content, performs sentiment analysis to get sentiment score, and indexes the needed posts information into Elasticsearch under structured indexes.



```
lizijing@lzj980517:~$ fission timer list
NAME           CRON       FUNCTION_NAME METHOD SUBPATH
mastodon-timer @every 30s mharvest      POST   /
reddit-timer   @every 3m  rharvest      POST   /
```

Figure 3. Fission Timer List

This event-driven architecture allows Fission functions to be invoked only when needed, eliminating the need for long-running backend services, which improves efficiency.

Meanwhile, the overall system follows a loosely coupled architecture where each component communicates via internal HTTP service calls. For example, Jupyter Notebooks sends HTTP GET requests to specific Fission routes, which in turn invoke the corresponding Fission functions that query Elasticsearch.

To securely manage access credentials for Elasticsearch, we created the "shared-data" yaml file to store the Elasticsearch username and password, and be included as a volume to all Fission functions that require Elasticsearch authentication. Each function reads credentials at runtime from the related file system path. This approach avoids the need to type sensitive information directly in source code and reduces duplication. Whenever the Elasticsearch credentials change, we only need to update the ConfigMap, without modifying every function code. Fission functions using Elasticsearch authentication communicate with ElasticSearch via the internal Kubernetes DNS.

To organize and manage internal clusters, it uses a modular namespace structure within Kubernetes. The "fission" namespace contains all fission components including timer trigger, routers, executor, and builders. And the "elastic" namespace is used for Elasticsearch cluster and Kibana dashboard. All system components mentioned are managed using declarative YAML files under the "spec" directory. These are deployed by executing the "fission spec apply --specdir specs --wait" command. This approach ensures easy reproducibility, version control and automation of updates.

Platforms and Tools

We used Kubernetes, Fission, ElasticSearch, Kibana, Jupyter Notebook, these core components to build our application. And Gitlab was used to share source code.

Kubernetes is the core of the infrastructure of this project, mainly responsible for the orchestration, resource scheduling, automatic expansion, and fault tolerance management of containerized applications. It provides cluster management and workload orchestration. With high scalability and elasticity, along with strong ecosystem support, Kubernetes is well-suited for managing large-scale distributed systems. That is why we use it.

As shown in Figure 4, on top of the Kubernetes platform, it introduced Fission, a serverless platform. Its use allows us to focus on writing function logic without having to deal with the configuration and management of the underlying infrastructure. It serves as a Function as a Service (FaaS) platform for processing data collection and analysis tasks. It simplifies the development process of REST API, and supports scheduled tasks and message queue integration. Nevertheless, troubleshooting is relatively difficult, and there are few reference materials available in the community, which increases the complexity of usage. We deployed a series of functions through Fission, which are mainly responsible for crawling and processing data from the outside, then storing the processing results in ElasticSearch, and completing the corresponding indexing operations.

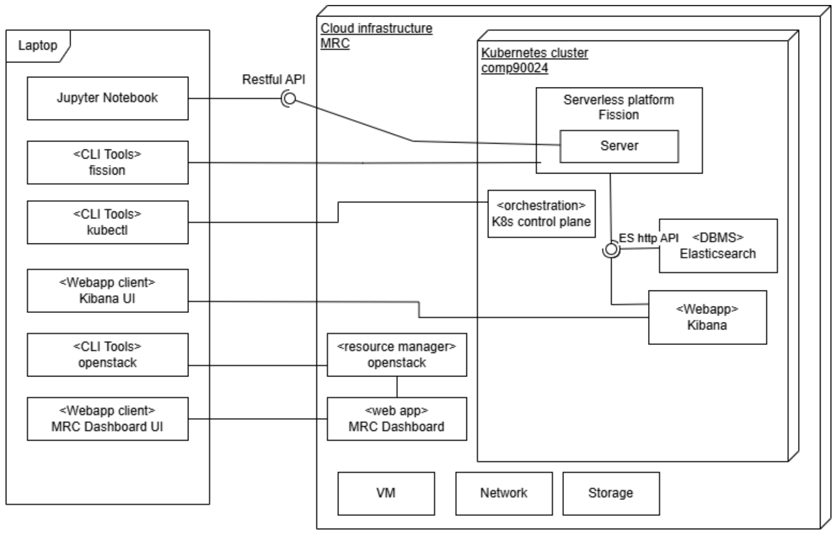


Figure 4. Infrastructure for this Project

ElasticSearch is the database for storing and querying social media data, mainly used for sentiment analysis and keyword tracking. As a powerful full-text search and data analysis engine, it offers strong query capability, fast indexing speed, and support for aggregation and filtering functions. So Elasticsearch is suitable for our streaming data analysis project.

To enable users to query and understand the ElasticSearch data more effectively, we integrated Kibana as a visualization tool. Kibana provides a graphical interface for users to query and visualize data results in ElasticSearch. As a data exploration and visualization dashboard, it features an intuitive interface, support for building visualization panels, time filtering, and queries. It has limited real-time interaction capabilities and places a heavy load on system resources, but it is enough for us.

Throughout the development process, we used GitLab to uniformly manage the source code, configuration files, and deployment scripts of the project, ensuring efficient version control and team collaboration. It serves as a source code repository and team collaboration platform. GitLab supports private repositories, merge requests, and issue tracking. However, its initial setup for SSH and permissions can be more complex than GitHub.

The front end of the system is implemented with Jupyter Notebook, which is mainly responsible for connecting to the RESTful API provided by Fission and visualizing the data queried from ElasticSearch. In this way, users can easily call the backend interface, obtain analysis results, and interactively explore in the form of charts, etc. As an interactive front-end dashboard, Jupyter Notebook is easy to use and supports dynamic charts and customized Python logic, which facilitates data analysis and display. However, it is not designed to build a complete web application, so it has certain limitations in interface interactivity and function expansion.

Except for the core components mentioned before, we also rely on a series of local tools to complete the construction and management of the system. OpenStack CLI was used to create the Kubernetes cluster, Fission CLI deploys functions, and kubectl CLI helped us view Kubernetes cluster status, the entire system forms a complete link from development, deployment, calling, and display.

Fission Harvesting and Query Functions

In this project, we collected and analyzed real-time social media data from Mastodon and Reddit platforms. These two platforms were accessed by their respective API clients, and we custom related harvester Fission functions running in our Kubernetes cluster. These functions are responsible for querying relevant content, doing sentiment analysis, and storing results in relevant Elasticsearch indexes.

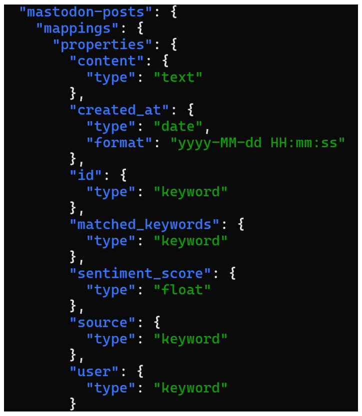
We also created a harvester function called “bluesky-au” to gather topic and Australia related posts from the Bluesky platform using the atproto Python client. This function was designed to search for scenario related keywords (e.g., “Trump”, “tariff”) and filter for Australian related keywords, analyze post sentiment, and index the results into Elasticsearch.

However, because of deployment issues and network configuration problems, we were not able to run this function successfully in the Fission environment. As a result, Bluesky data is not included in the final analysis pipeline.

Mastodon Live Data Harvesting

To access Mastodon, we used the “Mastodon.py” library to connect to the mastodon.au instance. The “mharvest” function collects new posts from Mastodon.au by querying the public timeline using the timeline(public) method with the “since_id” parameter, which retrieves only posts with IDs greater than it. We created a keyword list relevant to our scenario including "trump" and "tariff”, which can filter collected posts based on the presence of these keywords. Matched posts are then harvested and further processed by a sentiment analysis function, which uses the Sentiment Processing Methods to compute the compound sentiment score. Sentiment scores are shown as indicators of the post’s overall polarity (positive or negative) from -1 to 1. Each processed post is cleaned via regex-based HTML tag stripping to remove embedded markup. As shown in Figure 4, the needed features are all indexed into Elasticsearch under the “mastodon-posts” index.

To ensure real-time data processing, we deployed this via a fission timer trigger, which is named as “mastodon-timer” and configured by using a YAML specification to invoke the function every 30 seconds, which effectively achieved stream processing in our cloud-based infrastructure. In addition, extensive logging was implemented to record post id, keyword matching status and ElasticSearch indexing result. This makes it easier to debug and ensure traceability in our environments.

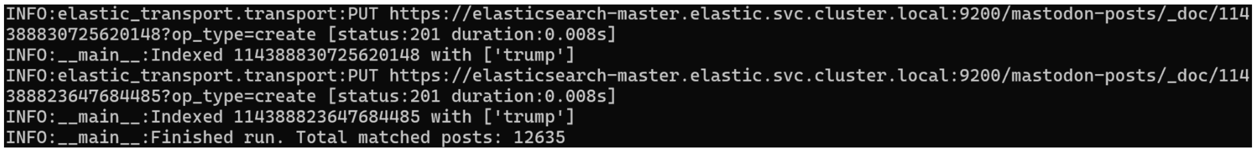


```
"mastodon-posts": {
  "mappings": {
    "properties": {
      "content": {
        "type": "text"
      },
      "created_at": {
        "type": "date",
        "format": "yyyy-MM-dd HH:mm:ss"
      },
      "id": {
        "type": "keyword"
      },
      "matched_keywords": {
        "type": "keyword"
      },
      "sentiment_score": {
        "type": "float"
      },
      "source": {
        "type": "keyword"
      },
      "user": {
        "type": "keyword"
      }
```

Figure 5. Index “mastodon-posts” Properties

Mastodon Past Data

The issue of Trump's tariff announcement is ongoing, as he announced the tariff policy on 3 April, the tariffs taking effect on 5 April, and the provisional tariff agreement between the US and China on 12 May. Because of Australia's strong trade ties with China, such US and China tariff developments are likely to have significant economic implications for Australia as well. Therefore, it is insufficient to just look at one point in time, but should focus on the process of change over time. Our group started this project at the end of April, so to get the data at the time of the announcement of the tariff policy we have to backtrack. The “pharvest-job.yaml” and “pharvest-scripts.yaml” are applied in our Kubernetes cluster. In the job, the latest id in the “mastodon-posts” is used as the starting point to extract data backwards. It is worthy mentioning that the post-id in Mastodon is unique, each post is indexed in Elasticsearch using its post id as the doc id. As a result, if the job found a post with the same id already exists, it will be overwritten, ensuring that only the most recent version is stored. Because of some technical limitations, we have to use “kubectl logs job/pharvest-job” to check the job’s log output and identify the latest index_id in the function records after each execution. Then, we updated it as the new max_id. and reapplied the new pharvest-job beginning at the new “max_id” again to collect the past data. For example, as shown in Figure 5, the latest index_id is “114388823647684485” in this running turn.



```
INFO:elastic_transport.transport:PUT https://elasticsearch-master.elastic.svc.cluster.local:9200/mastodon-posts/_doc/114
388830725620148?op_type=create [status:201 duration:0.008s]
INFO:__main__:Indexed 114388830725620148 with ['trump']
INFO:elastic_transport.transport:PUT https://elasticsearch-master.elastic.svc.cluster.local:9200/mastodon-posts/_doc/114
388823647684485?op_type=create [status:201 duration:0.008s]
INFO:__main__:Indexed 114388823647684485 with ['trump']
INFO:__main__:Finished run. Total matched posts: 12635
```

Figure 6. “pharvest” function running log

In order to get more and more data, we used these yaml files to run the past data harvester job and keep it running until it successfully collected the data on 3rd April. However, this job took a lot of time and can be blocked under multiple events such as the temporarily unavailable Mastodon server. It can be improved by deploying it as a Fission function and using a Fission timer.

Reddit Live Data Harvesting

To gather topic-specific material from Reddit, we developed a Python function called rharvest. This function accesses the r/australia subreddit using the PRAW (Python Reddit API Wrapper) client and retrieves the most recent 50 public posts on each run. A predefined keyword list (e.g., “Trump”, “tariff”) is utilized to select posts on political or economic issues. The title and text of each post are concatenated and cleaned by removing all the HTML tags.To enable real-time sentiment tracking, we added a timer-based harvesting capability for Reddit. The “rharvest" function is executed as a Fission function based on the specification in function-rharvest.yaml and is scheduled to run every 3 minutes as a TimeTrigger defined in timetrigger-reddit-timer.yaml.

To prevent reposting of posts which have already been seen, the function has a since_id mechanism in place via a separate Elasticsearch index (reddit-lastid) to store the ID of the last processed post. In each run, the harvester omits all posts which have been older than the saved ID and saves the ID at execution completion.

For every matched post, the function computes the sentiment score with our proprietary analyzer, constructs a document consisting of metadata like ID, author, content and timestamp, and indexes it into the reddit-posts index within Elasticsearch. This makes for efficient, stateful, and consistent ingestion of the data for subsequent analytics.

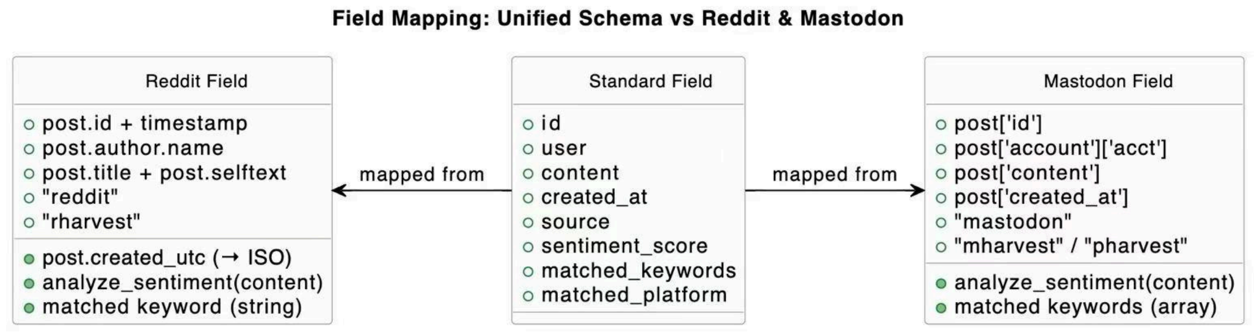


Figure 7. Unified Data Field Mapping from Reddit and Mastodon to Standard Schema

Bluesky Live Data Harvesting

In order to investigate new platforms beyond social media in the classical sense, we introduced the harvester function bluesky-au for the Bluesky platform using the atproto Python client. This function was created to conduct keyword-based searches on public posts for specific words (e.g., “Trump”, “tariff”) and also for Australia-specific terms like “Melbourne”, “aussie”, or “sydney”. The harvester logs in via an app password, searches for up to 100 posts on each page for each keyword, and paginates through as many as 100 pages in each run. On each match post, the function pulls out the content, handle of author, and timestamp, does sentiment analysis using our in-house model, and indexes the outcome into the “bluesky-posts” index in Elasticsearch. To be relevant, only posts meeting both topic-specific and region-specific criteria are processed.

While we had implemented and run the function locally successfully, we did not manage to deploy it to the Fission environment successfully due to unsolved runtime or container-level networking problems. Therefore, Bluesky data was not part of the final aggregated analysis pipeline.

Sentiment Processing Method

We calculate each post’s sentiment using the VADER (Valence Aware Dictionary and Sentiment Reasoner) library, which is specifically tuned for short, informal texts like social-media messages. After stripping out URLs, mentions, and formatting from the raw text and normalizing it to lowercase, VADER analyzes each word against its built-in sentiment lexicon to produce four component scores—positive, negative, neutral and a single “compound” score that aggregates them into a value between –1.0 (extremely negative) and +1.0 (extremely positive). By convention, compound values at or above +0.05 are labeled “positive,” values at or below –0.05 are labeled “negative,” and anything in between is treated as “neutral.” Then, we set invalid or abnormal sentiment score as 2.0. These thresholds were empirically validated by VADER’s authors[4] on a large corpora of social posts to minimize misclassification of subtly worded content. We record both the raw component scores and the derived label for each document in Elasticsearch so that our aggregations can compute time-series trends of the average compound score and also count how many posts fall into each sentiment category. Because the compound score reflects the overall balance of positive and negative valence in a post, it drives our visualizations of daily and weekly sentiment dynamics and ensures consistent, interpretable insights.

Elasticsearch Query Function

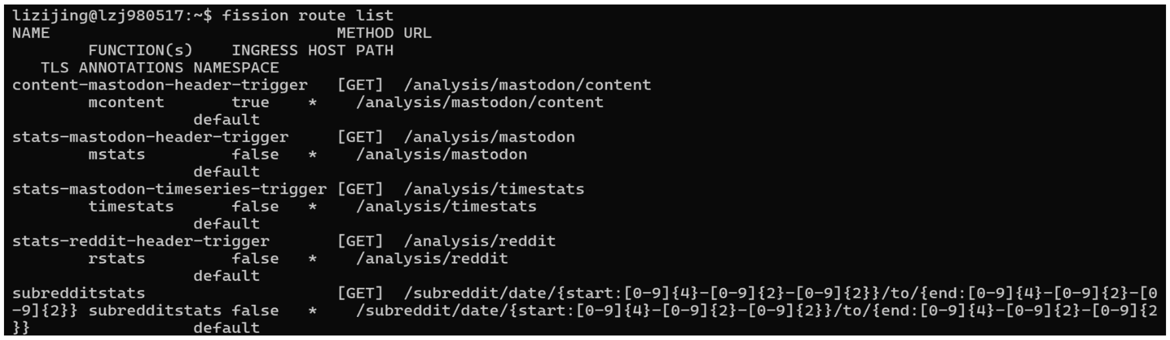
Our system integrates five Elasticsearch-based query functions: “mcontent”, “mstats”, “rstats”, “timestats”, and “subredditstats”. Among them, “mcontent” is the most unique - it does not perform aggregation or filtering operations, but directly extracts the most positive and negative individual posts from the Mastodon dataset based on sentiment scores, providing an intuitive qualitative analysis perspective through specific content. The remaining four functions are used for aggregated sentiment analysis, mainly returning summary statistics such as the number of posts and average sentiment scores. Among them, “mstats” and “rstats” perform keyword-level aggregate analysis on the Mastodon and Reddit platforms respectively; “timestats” supports multi-keyword queries and can set exclusion conditions, which is suitable for analyzing the trend of sentiment changes in the time dimension; and “subredditstats” aggregates Reddit data by sub-forum and time range, which helps to explore the structural discussion characteristics within the platform. These five functions complement each other, supporting both cross-platform quantitative trend analysis and helping users to deeply understand the emotional expression on social media from specific content.

Front-End Interface & User Experience

This section describes our interactive front-end interface based on Jupyter Notebook. The frontend system is not only used to visualize analysis results, but also allows users to query the real-time data in

Elasticsearch via RESTful APIs. The Jupyter notebook communicates with Fission functions deployed on Kubernetes via HTTP requests through Fission Routes to dynamically retrieve and display the sentiment analysis results. In order to meet different analysis needs, we define several functions-related Fission Routes, each corresponding to a specific backend function as mentioned before.

These Fission Routes ultimately connect to different Fission functions which search and do statistical analysis for data from correlated Elasticsearch indexes, such as mastodon-posts and reddit-posts, and return structured JSON data results. The front-end Jupyter Notebook receives this data and uses the visualisation functions in plot_utils.py for graphical presentation and analysis.



```
lizijing@lzj980517:~$ fission route list
NAME                              METHOD URL
        FUNCTION(s)    INGRESS HOST PATH
   TLS ANNOTATIONS NAMESPACE
content-mastodon-header-trigger   [GET]  /analysis/mastodon/content
        mcontent       true    *    /analysis/mastodon/content
                   default
stats-mastodon-header-trigger     [GET]  /analysis/mastodon
        mstats         false   *    /analysis/mastodon
                   default
stats-mastodon-timeseries-trigger [GET]  /analysis/timestats
        timestats      false   *    /analysis/timestats
                   default
stats-reddit-header-trigger       [GET]  /analysis/reddit
        rstats         false   *    /analysis/reddit
                   default
subredditstats                    [GET]  /subreddit/date/{start:[0-9]{4}-[0-9]{2}-[0-9]{2}}/to/{end:[0-9]{4}-[0-9]{2}-[0
-9]{2}} subredditstats false   *    /subreddit/date/{start:[0-9]{4}-[0-9]{2}-[0-9]{2}}/to/{end:[0-9]{4}-[0-9]{2}-[0-9]{2
}}                 default
```

Figure 8. Fission Route List

Overview

Jupyter Notebook is primarily used to analyze changes in public opinion regarding Trump and tariff on the Mastodon.au and Reddit platforms. In this project, "api_utils.py" is responsible for retrieving processed data from Elasticsearch via API endpoints connected to Fission functions. It supports flexible configuration of parameters such as keywords, timeframes, platforms, and returns structured sentiment and posting statistics.This raw data is processed by multiple functions in "plot_utils.py", which convert the data returned from "api_utils.py" into various visualizations, including sentiment trend charts, posting volume line charts, keyword bubble charts, sentiment comparison bar charts, dynamic trend animation charts and geographic distribution maps. And the jupyter notebook file "trump_tariff_analysis.ipynb" is the frontend dashboard for our project, which sequentially loads the data, generates charts, and interprets and analyses them in relation to key time nodes (e.g. Trump's announcement of tariffs on April 2, 2025 as shown in [2]) by calling the functions of the two modules mentioned above. This notebook ultimately outputs images and conclusions with analytical value. Overall, these three files work together to achieve a complete process of data collection through Fission routers & structured processing → multidimensional visualisation → dashboard, helping users to gain insight into the intensity of Trump-related discussions across platforms, their emotional attitudes, and their changes with the advancement of events.

Our Jupyter notebook dashboard analyzes three main sections: Mastodon discussion, Reddit discussion, and cross-platform discussion. In the first part, we analysed the discussion about Trump and Tariff on the Mastodon platform by Australian users. The second part of the analysis focuses on Reddit, comparing the post volume and sentiment scores across different regions (e.g. Brisbane, Perth). The last part compares the number of posts and sentiment scores for the topics of Trump and tariffs across both Mastodon and Reddit platforms. The data analysis component is introduced in the Analytical Scenarios and Visualization section in our report.

Supported Fission Routes

Jupyter notebooks support rich visualisation capabilities using libraries such as matplotlib and seaborn. It allows users to dynamically query data and immediately view the results in a graphical format. These graphs are in real time, updating at any time as more data is captured.

Of all functions that we have deployed in our system, only a subset were actually rolled out with active triggers of type HTTP. They include Mastodon and Reddit sentiment endpoints, exposed as Fission functions and invoked using HTTP GET requests. For instance, functions mstats and rstats have support for on-demand aggregation of sentiment by date range and keywords, which is generally invoked by remote clients like Jupyter notebooks or curl scripts using custom headers. The system also offers endpoints to retrieve matched Mastodon posts (mcontent), Reddit aggregation at the subreddit level (subredditstats), and time series of sentiment (timestats). This design keeps the ingestion layer (e.g., mharvest, rharvest) separated from the API querying layer, enhancing modularity as well as runtime efficiency.

Because of varying structures in the field of the harvested data for each platform, our Elasticsearch queries operate on differing keyword fields for each source index.For Mastodon, we have stored the keywords in field match keyword and can use it as is for aggregating on terms.For Reddit, we store the matching keywords as a text field in the form of a list in matched_keywords. To allow for aggregation on it, we query the .keyword subfield (matched_keywords.keyword), which leaves us with the full string untouched. No tokenization is done on it. Effective keyword-level sentiment analysis on both sources is facilitated by this distinction.

Use of visualization tools

In this project, we applied a wide range of visualisation tools to support users' interactive analysis and understanding of Trump-related social media sentiment data. The main tools used include Plotly, Folium, and Pandas, which are integrated into Jupyter Notebook. Restful APIs to fetch query results from Elasticsearch indexes to generate dynamic charts and interactive views. The analysis visualizations are shown in Analytical Scenarios and Visualization Section.

1. Line Chart: We use Plotly to implement a dynamic line chart that shows the trend of average sentiment score over time. Users can go back to any point in time with a sliding bar to examine how sentiment fluctuated before and after key events such as "tariff announcement", "policy delay", "tariff cut", etc.
2. Bubble Chart: It shows the relationship between the sentiment scores and the keyword frequencies on a given day. The horizontal axis represents average sentiment, the vertical axis shows the number of occurrences, and size of the bubble indicates the amount of frequency, and the colours represent different keywords. This chart helps to compare the differences in sentiment and heat across multiple keywords.
3. Animated Trend: The animation function of Plotly is used to implement a sentiment timeline graph, which dynamically visualizes the public opinion shifts on Trump-related topics as key events unfold. This enhances both the narrative quality and interactivity of the visualisation.

4. Map: Under the data conditions where geographic information exists, we use Folium to map the geographic distribution of emotions. These visualizations show the attitudes of different regions towards the relevant events, this also supports spatial level analyses.
5. Statistical Tables: uses Pandas to generate keyword sentiment statistical tables, including average scores, number of occurrences, time intervals, and other information, which is easy for developers to debug and user-friendly to browse the data results.
6. These charts and interactive visualization greatly enhance the user experience, enabling even non-technical users to intuitively understand the trends and meaning behind the data. Additionally, all visualizations can be updated in real-time via URL queries, ensuring up-to-date and accurate results.

Access Instructions

The front end of this system is implemented in Jupyter Notebook, which provides an interactive interface for data visualization and analysis. Users can interact with the backend services by calling predefined API functions encapsulated in a separate module "api_utils.py". This approach improves the modularity of the system and enhances the readability and maintainability of the Notebook.

With these interfaces, users can access RESTful APIs deployed on Kubernetes and Fission platforms to dynamically obtain and display analysis results, including data tables, sentiment analysis graphs, and bar charts. The Notebook introduces practical functions such as "get_mastodon_posts_counts", "get_trump_post_volume_on_day", and "get_daily_sentiment_count", which greatly simplifies the process of obtaining and processing social media data. In addition, users can also interactively explore the trend changes and sentiment fluctuations of keywords, and gain insight into the direction of public opinion in a specific time period.

Before running the Notebook, ensure that all backend services (including Fission functions and ElasticSearch indexes) are properly deployed and running. Once the preparations are complete, users can open the "trump_tariff_analysis.ipynb" file in Jupyter Notebook environments and execute the cells in order. The system will automatically fetch and visualize the latest data from Mastodon and Reddit platforms, allowing for interactive analysis of the public opinion dynamics of discussions related to Trump and his tariff policies.

Deployment and Scalability

This section focuses on the deployment process and scalability strategies implemented in our cloud environment.

Our system was deployed on the NeCTAR Research Cloud using a Kubernetes cluster with one master node and three worker nodes by using the "openstack coe cluster create" command. And we can check the cluster health status by using the "openstack coe cluster show" command. This setup achieves a balance between resource availability and operational simplicity. To maintain a fixed node count, the Kubernetes scheduler dynamically manages pod distribution based on node health and capacity, enabling internal scalability across system components.

All system services were not deployed one by one. Instead, we implemented various tools to create, update and manage them. We managed Elasticsearch and Kibana by using Helm charts, which allowed configuration of replica count, storage size and namespace isolation. This made these two services easy to reproduce and reconfigure across different environments. We set the YAML specification to manage all functions, whether they are newly created or being redeployed. All functions, environments, routers and timers were defined in YAML files under the "specs" directory. When we implemented "fission spec apply --specdir specs --wait", it would deploy those and update functions in a consistent and automated manner, provided there were no errors. This approach ensured that every function could be versioned or scaled independently.

Our system supports horizontal scalability by design. New data harvesters for different platforms can be added by simply creating new Fission functions and defining corresponding time triggers. All functions can run in parallel without interfering with each other and are event-driven, thus additional instances can be scheduled concurrently without conflict. For example, "mharvest" function and "rharvest" function can run in parallel with independent timers, and the analytics functions such as "mstats" and "rstats" can be invoked concurrently on-demand by the frontend.

Although we did not use full autoscaling (e.g., KEDA[6]) in this project, our system is ready for future dynamic scaling. Kubernetes natively supports replica sets, it allows configure different replicas of a function to run and Kubernetes would automatically schedule them. Meanwhile, Fission also has compatibility with scale-out strategies (such as Redis queue-based triggers), which provides a strong foundation for handling large data volumes or traffic bursts.

In summary, our deployment strategy prioritizes modularity, reproducibility, and scalability. By using Helm, Fission specs, and a loosely coupled architecture, we can apply rapid function deployment, isolated failure domains, and clear pathways for horizontal expansion of both data sources and analytics endpoints.

To apply automation and reproducibility, we set up a GitLab CI/CD pipeline to automatically implement the latest Fission specs and push changes to the deployment environment. As shown in Figure 7, the pipeline consists of a deploy stage that applies Fission specs, followed by a verify stage that checks whether functions were deployed successfully and whether data ingestion is functioning as expected (verify-es-ingest is limited due to disconnection to the internal services of the cluster). When any changes are committed to the gitlab repository, the pipeline would validate the YAML files again and execute the "fission spec apply --specdir specs --wait" command. This ensures that the cluster stays synchronized with the version-controlled source, reduces manual deployment effort, and prevents configuration drift.

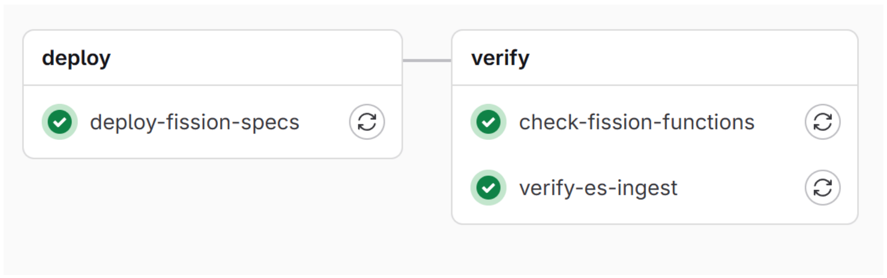


Figure 9. GitLab CI/CD Pipeline Structure

Analytical Scenarios and Visualization

Scenario: What Do Australians Think of Donald Trump and Tariffs?

Donald Trump’s implementation of tariffs has exerted a profound influence on global trade patterns. As an economy heavily reliant on exports, Australia feels the direct impact of these policy shifts. To understand how these changes resonate within Australian society, our project explores two complementary social-media ecosystems: Mastodon (specifically its Australia-focused instances) and the r/Australia subreddit. While Reddit offers a more centralized forum where public discourse can be rapid and highly visible, Mastodon’s decentralized architecture often surfaces niche and underrepresented viewpoints. By collecting and analyzing posts from both platforms, we gain a fuller picture of sentiment around Trump’s tariff agenda and can capture fluctuations in opinion that traditional surveys or single-platform analyses might overlook.

Our analysis begins by leveraging Mastodon’s unique, decentralized data to illuminate international political perceptions in ways that more centralized platforms cannot. First, we compare the average sentiment scores for posts about Donald Trump and tariffs on Mastodon with those on Reddit over the same period; this comparison not only highlights differences in tone between communities but also underscores Mastodon’s capacity to surface niche viewpoints that might be muted elsewhere. Next, we trace the daily flow of key terms such as “Donald Trump,” “trade war,” and “tariffs” on both platforms. By using keywords‑count histograms enhanced by average sentiment, we reveal how Mastodon conversations spike around pivotal events and how their emotional tenor evolves over time. Finally, by breaking posts into hourly buckets, we map intraday peaks in discussion volume and sentiment on Mastodon, demonstrating when Australians most actively engage with these topics and with what emotional valence. Collectively, these steps not only show what Australians are saying about Trump and tariffs, but also showcase Mastodon’s singular role in shaping and reflecting the broader narrative of international political discourse.

Visualization & Insights

Figure 10 is a dynamic line graph and is in real time, updating at any time as more data is captured. By dragging the slider at the bottom of the chart, we can observe how sentiment scores evolve over time. Before the tariffs were announced, the sentiment score was already in negative territory. Following the tariff announcement, negative sentiment increased significantly and persisted for a period of time, reflecting a surge in criticism, concern, and opposition on social media. The news[2] of the 90-day delay had some positive impact, with negative sentiment beginning to ease. Users may have interpreted this move as a temporary concession rather than a policy reversal. News of Trump's agreement to cut tariffs had a noticeable positive impact on user sentiment, with negative sentiment abating significantly. However, by mid-May, negative sentiment appeared to pick up again. This suggests that while users initially responded positively to this policy change, they generally maintained a cautious, critical attitude.

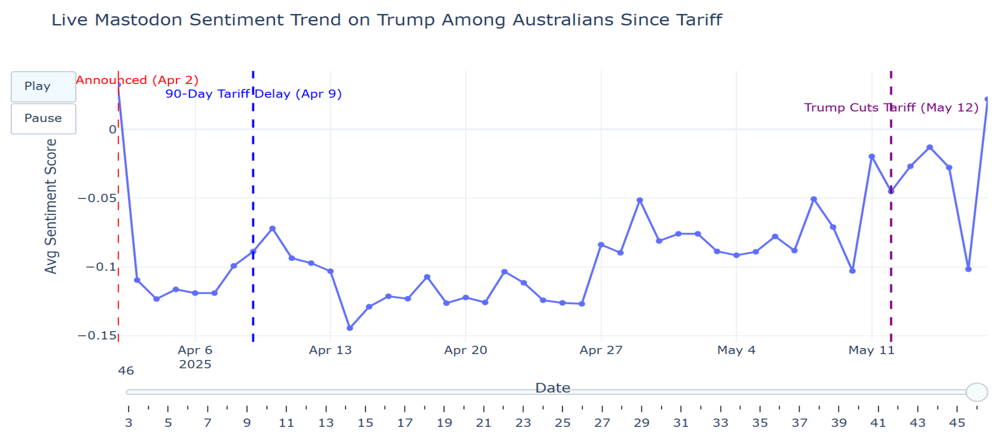


Figure 10. Live Mastodon Sentiment Trend on Trump on Mastodon.au

Following the announcement of the tariffs, in Figure 11, there was a noticeable increase in the number of posts related to Trump, which reached a peak in a short period of time. This indicates that the announcement of the tariffs policy generated significant interest and discussion among Mastodon users in Australia. After Trump agreed to cut the tariffs, we can observe another notable rise in the number of posts, followed by a rapid decline. This suggests that the news of Trump agreeing to reduce tariffs also drew attention from users, but the duration of this discussion did not last as long as it did when the tariffs were initially announced. Overall, this chart demonstrates that major tariff-related events greatly influenced the volume of discussion. The tariff announcement triggered the strongest spike in discussion, while the agreement to cut tariffs also generated a smaller and shorter surge in discussion. Between these major events, the amount of discussion about Trump declined, but there was still a certain amount of daily discussion.

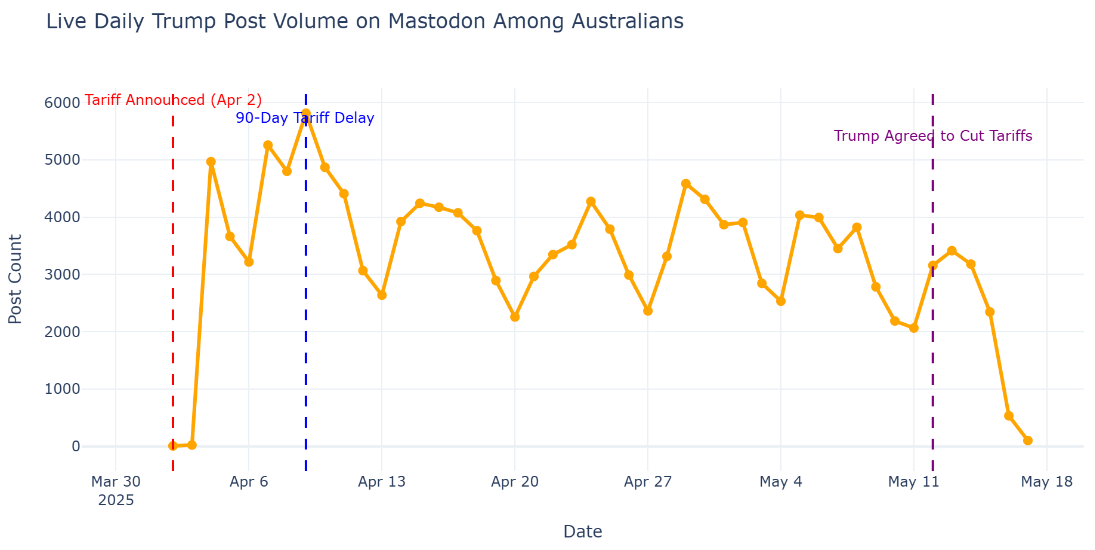


Figure 11. Live Daily Trump Post Volume on Mastodon Among Autralians

In Figure 12, the integrated plot of count of posts and average sentiment, we can see the number of posts increased significantly after April 2, which is consistent with previous analysis and suggests that the event triggered a lot of discussion. The average sentiment dropped sharply after the tariffs were announced, falling into negative territory and remaining low for some time. This suggests a broadly negative attitude towards Trump's tariff announcement. Post volume fluctuates slightly after the announcement of the 90-day tariff delay, but no significant spikes. Average sentiment recovered slightly but remained in the negative territory. This suggests that the news of the delay had minimal impact on overall sentiment. There was another noticeable uptick in discussion after Trump agreed to cut tariffs on May 12, but this was followed by a rapid decline. Public sentiment of this topic showed a significant rebound, at one point approaching or even briefly exceeding 0, suggesting an improvement in user attitudes. However, this positive change did not last long and then fell back into negative territory.

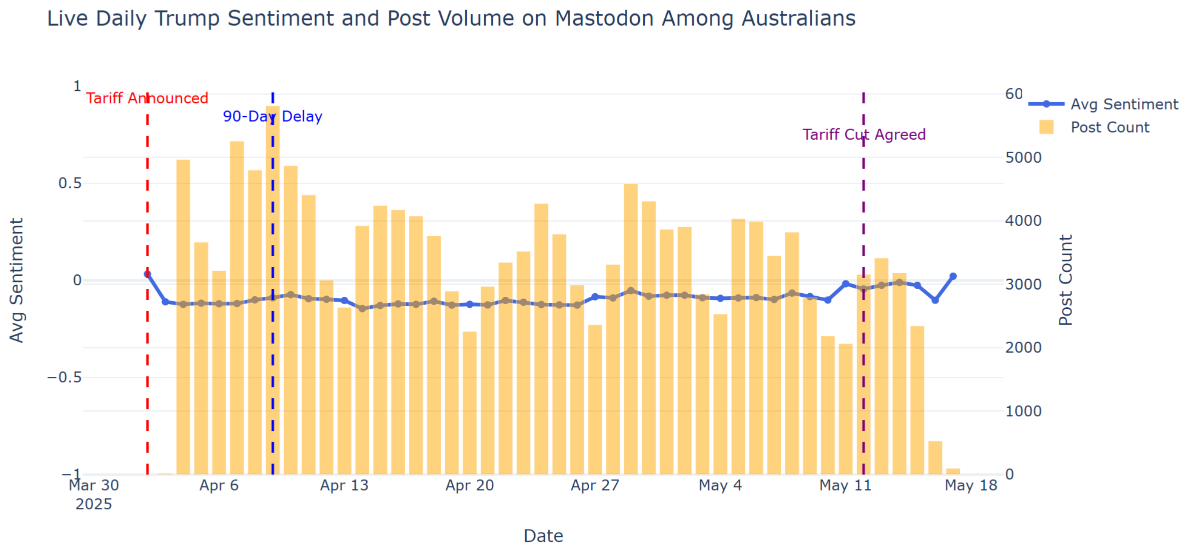


Figure 12. Live Daily Trump Sentiment and Post Volume on Mastodon Among Australians

Figure 13 shows trends in the number of posts on Mastodon for various Trump-related keywords since the tariffs were announced in early April. The use of the keyword "Trump" is overwhelmingly dominant, peaking at over 6,000 posts per day. In contrast, "Donald Trump" appears far less frequently, indicating that users tend to prefer shorter forms in informal discussions. Other keywords, such as "Trumpism" and "Trumpian" are nearly negligible in comparison.

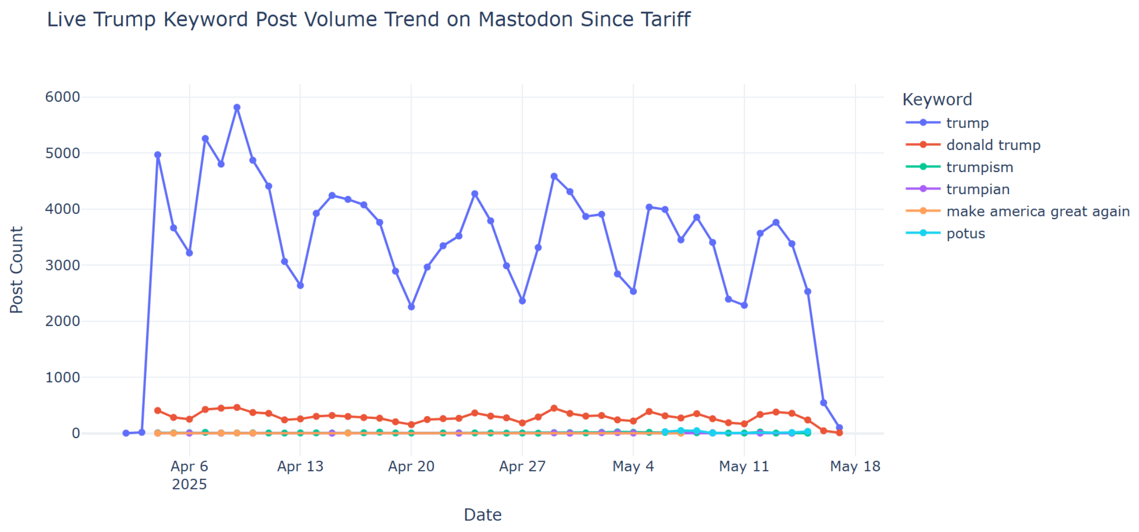


Figure 13. Live Trump Keyword Post Volume Trend on Mastodon Since Tariff

In Figure 14, the amount of discussion about Trump is much higher than the amount of discussion about tariffs. The number of posts related to Trump (168,957) is more than ten times the number of posts related to tariffs (12,135). This suggests that the topic of Trump dominates the discussion, while tariffs remain a sub-topic, which may indicate that the public is more interested in Trump's overall behaviour, rhetoric or persona than in a single economic policy issue.

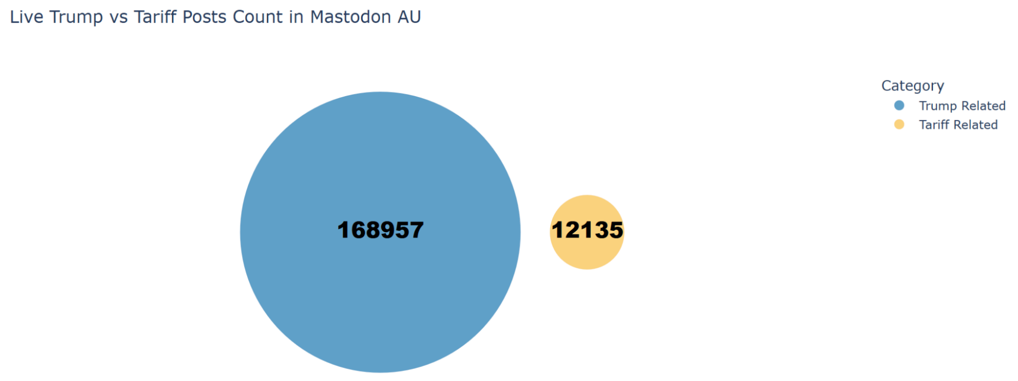


Figure 14. Live “Trump” VS. “Tariff” Posts Count on Mastodon

Figure 15 is a time-series line graph, with the x-axis representing the dates (from early April to mid-May 2025) and the y-axis being the average sentiment score. Each coloured line represents a different Trump-related keyword. Overall, most Trump-related keywords show a negative sentiment bias. The pattern of sentiment fluctuation varies across keywords, for example, "tariff" shows a more dramatic change in sentiment, while "trade war" is consistently negative. Some keywords (e.g., "maga", "potus", and "Trump") show a short-lived rebound in sentiment in mid-to-late April and early May, while "make america great again" and "trade war" maintain more negative sentiment levels. The limited number of data points for "make america great again" and "Trumpian" suggests that the volume of discussion of these keywords is relatively low.

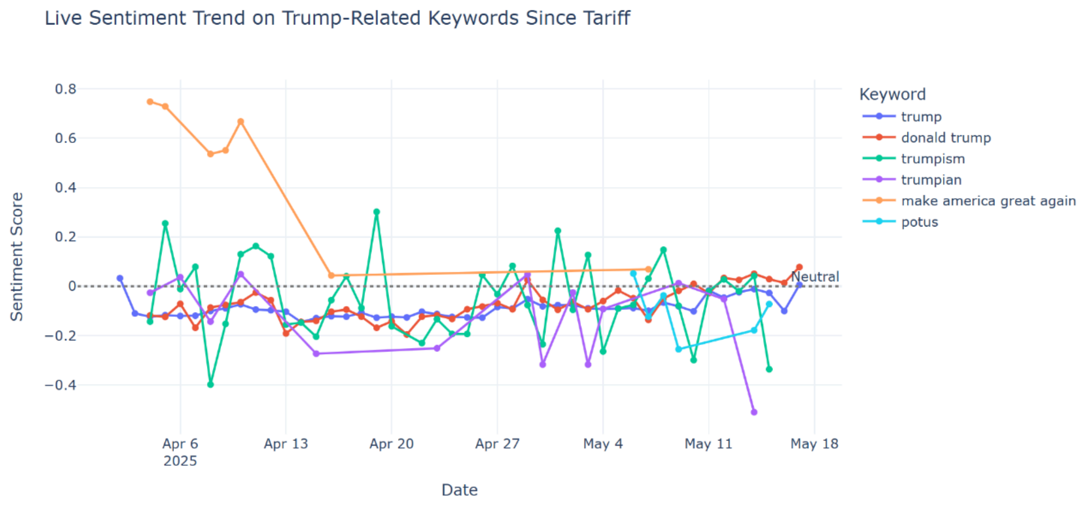


Figure 15. Live Sentiment Trend on Trump-Related Keywords Since Tariff

Figure 16 shows the average sentiment comparison of users on Reddit (blue) and Mastodon (green) on topics related to Trump and his tariff policy. The results show that there are obvious differences in the attitudes of the two platforms towards Trump himself: Reddit users generally express positive emotions (+0.31), while Mastodon users tend to be negative (-0.09), reflecting the significant differences in political orientations of communities on different platforms. On topics related to tariffs, the sentiment of both platforms is slightly negative, with Reddit at -0.07 and Mastodon at -0.05, indicating that although users have different views on Trump himself, they hold a relatively consistent critical attitude on the issue of tariffs. Overall, the discussion around Trump is more polarized, while the topic of tariffs has triggered relatively mild and convergent negative emotions.

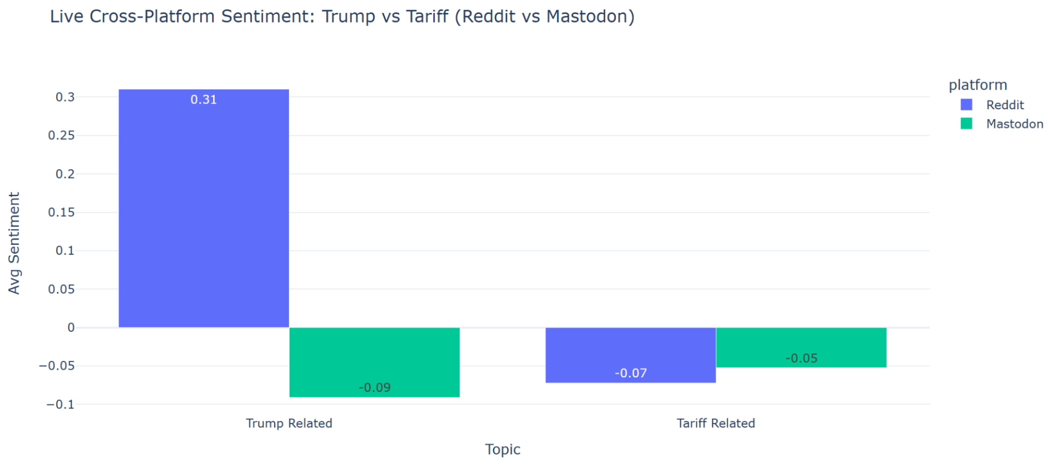


Figure 16. Live Cross-Platform Sentiment Comparison

Figure 17 shows the changes in the number of posts related to Trump and tariffs, counted every 3 hours from May 5 to 11, 2025. Overall, the number of posts from May 6 to 10 remained high, with obvious peaks in the noon and evening hours, indicating that users were more engaged during these time periods. Among them, May 6 and 9 were the most active, which may be related to major news events or official announcements. In contrast, the number of posts on May 5 and 11 was significantly lower, especially on weekends, which may be affected by the "weekend effect" or topic fatigue. This trend shows that public discussions are mainly concentrated on weekdays.

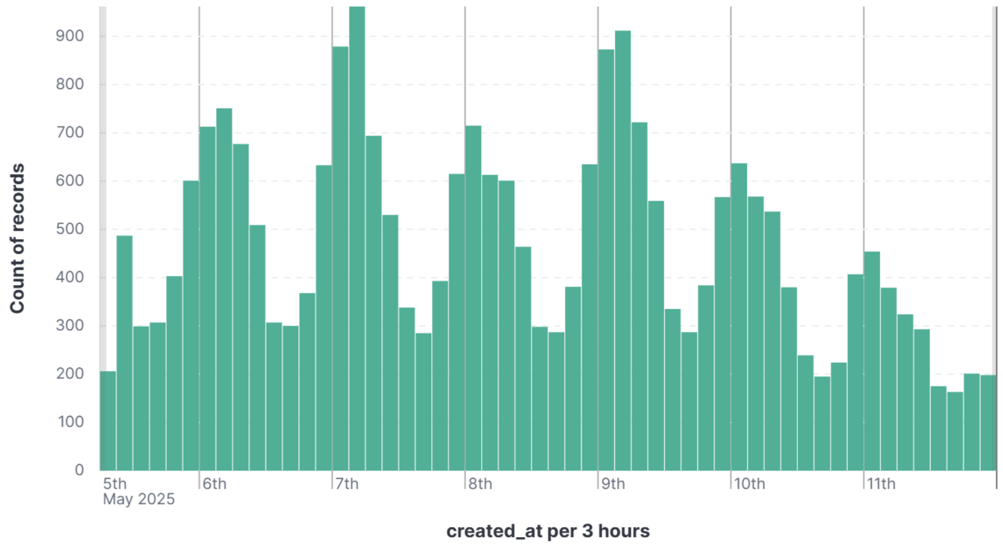


Figure 17. Post Count Trend Over One Week (From Kibana)

In Figure 18, green marks indicate positive sentiment, while red marks represent negative sentiment towards Donald Trump and his tariff policies. Brisbane is shown in green, indicating that the local public is relatively supportive or optimistic, which may be related to news such as tariff extensions or reductions. Sydney and Perth are marked in red, reflecting obvious negative sentiment, which may stem from people's concerns about the economic impact of these trade-oriented cities. In addition, a light green cluster appears in central Australia, showing a slightly positive view, indicating that the public attitude in inland areas is more neutral or relatively less affected by tariffs.

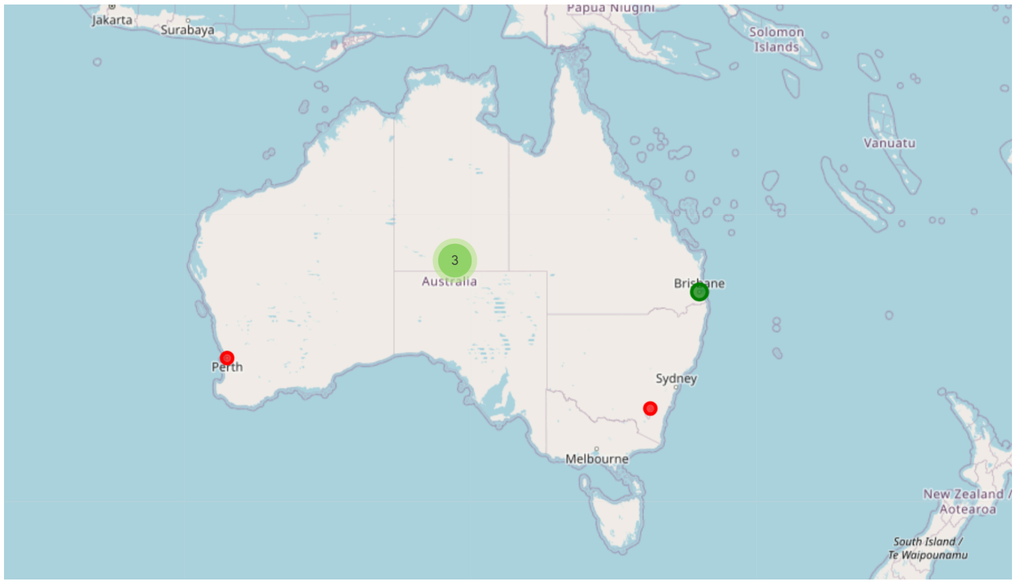


Figure 18. Post Volume and Sentiment across Australia on Reddit

Summary of Analysis

In summary, our analysis has successfully demonstrated that Australians' online discourse around Donald Trump and tariff policy differs markedly not only across platforms but also over time and by topic focus, thereby validating our initial hypotheses. By comparing aggregate sentiment scores we confirmed that Reddit users are, on average, more positive toward Trump (+0.31) than Mastodon users (–0.09). However, both communities share a mild critical stance on tariffs, reinforcing our hypothesis that platform culture shapes political expression. The daily keyword‑count histograms and average‑sentiment trends clearly captured the pronounced spikes in discussion and corresponding mood swings which are mainly triggered by major policy announcements (tariff imposition, 90-day delay, tariff rollback), affirming our belief that discrete events drive engagement surges and sentiment volatility. Finally, our hourly aggregation exposed consistent peaks in posting volume and sentiment fluctuations around midday and evening hours, confirming that user attention and emotional response follow predictable intraday patterns. Together, these findings underscore the power of cross‑platform, time‑aware sentiment analytics. Our Fission–Elasticsearch pipeline shows Australian public opinion on Donald Trump and his international trade policies with the granularity and responsiveness necessary to verify even subtle conjectures about when, where, and how strongly these views are expressed.

Error Handling & Fault Tolerance

Throughout the development and deployment of our system, we encountered multiple technical challenges and limitations ranging from low-level implementation errors to platform-specific limitations and deployment inconsistencies. This section illustrates these issues and the solutions or mitigations we applied.

Technical Level

1: Sentiment Score Format Incompatibility

The initial sentiment processing Method we mentioned before returned results in a dictionary format, which contains positive, neutral, negative and component scores. While this format was rich in information, it was not compatible with Elasticsearch when we attempted to store this kind of result in Elasticsearch indexes, which resulted in a mapping error. To resolve this, we simplified the sentiment processing method output by extracting only the compound score as a float value. Then, this result could be successfully input into Elasticsearch.

2: Inadequate Timer Frequency for Data Collection

At the beginning of the YAML specification part, we set the "mharvest" function timer to 3 minutes (make one attempt to run the function every 3 minutes), with only hundreds of Mastodon posts per day. But when we directly used the "fission fn test" command to test this Fission function, we observed that it could harvest approximately 1-2 relevant posts in 20 seconds, which means up to 6-12 related posts might be published over a 3-minute span. However, with the timer originally set to trigger only once every 3 minutes, the function was able to only collect posts from just a single moment in that interval. So the setting of 3 minutes caused our "mharvest" function to miss a large portion of relevant data, which makes us miss many others that were published in between. After reducing the timer interval to 30 seconds, the data volume increased significantly.

3: Flask Version Incompatibility in Kubernetes

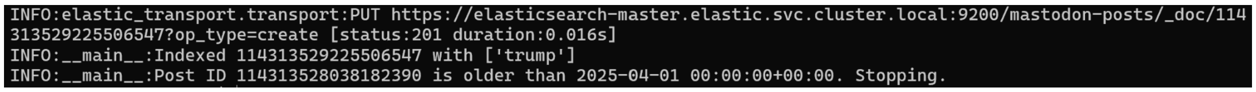
In the requirements.txt file, the version of Flask was initially set as 2.8. However, when we deployed the package with this version, it showed "fail" status for creating the package. We investigated and found that Kubernates' environment does not support this version of Flask. So we downgrade the Flask version to 2.2.5 in requirements.txt file, to resolve this issue and allow packages to create successfully within the cluster.

4: Duplicate Post Storage Due to doc_id Format

Initially, the doc_id parameter was set to a combination of posts id and timestamp. However, an issue occured where some Mastodon users post the same contents twice with identical post id but different timestamp, likely due to network issues. For example, if a user posts "I like Trump" at 12:11:05 and again at 12:11:07, both instances would be collected by our functions as two pieces. This is undesirable, as each post id is unique for Mastodon. To resolve this, the doc_id was updated to use only the post id. If duplicate posts are encountered again, they will overwrite the previous entry instead of being stored twice.

8: Incorrect Boundary Detection in Past-Data Harvester

When using the past function to collect data, the "stop_date" was initially set to April 2, 2025, the day Trump announced the tariffs. However, it was observed later that the function sometimes unexpectedly pulled records from earlier dates as shown in Figure 19. For example, even though most of the data was around April 8, the function retrieved one record with an ID indicating a date earlier than April 2, causing the program to assume that it had exceeded the boundary and terminate the crawling process. To solve this problem, the stop_date was adjusted to March 20, 2025 to ensure that the required time range is fully covered.



```
INFO:elastic_transport.transport:PUT https://elasticsearch-master.elastic.svc.cluster.local:9200/mastodon-posts/_doc/114
313529225506547?op_type=create [status:201 duration:0.016s]
INFO:__main__:Indexed 114313529225506547 with ['trump']
INFO:__main__:Post ID 114313528038182390 is older than 2025-04-01 00:00:00+00:00. Stopping.
```

Figure 19. “Pharvest-job” Log

Platform-specific Issues and Solutions

1: Since ID Mechanism Causing Data Skips

The since_id mechanism skipped older posts during testing, and was fixed by adding a test mode that skips the since_id check for full backfill history.

2: Sentiment Analyzer Crashes on Edge Cases

The sentiment analyzer was crashing on malformed, non-English, or symbol-laden text, which was addressed by enclosing the analysis in a try-except block that settles on a neutral score on failure.

3: Unicode and Encoding Issues

Special characters (such as emojis and CJK scripts) produced encoding issues, which were overcome by normalizing all the strings into UTF-8 with errors="ignore" for logging and indexing.

4: Reddit API Limitation on Historical Data Retrieval

The Reddit API from PRAW constraints .search() and .new() to only return the last 1000 most recent posts for each query. This limitation renders it impossible to reliably gather long-term history through local execution. This limitation can't be circumvented in testing on-premise. To record full history across time, the harvester should be installed in the cloud infrastructure and executed continuously or scheduled (e.g., through Fission timers) in order for information to be processed bit by bit as new posts are added.

5: GitLab CI Cannot Access Cluster-Internal Services

One issue for GitLab CI was its “verify-es-ingest” job cannot run, when it attempted to check whether the Fission function successfully harvests posts into the Elasticsearch index. It tried to use the Elasticsearch address “https://elasticsearch-master.elastic.svc.cluster.local:9200” to connect to Elasticsearch, but it failed. Because CI runners are outside the cluster, they cannot reach our internal services. To handle this gracefully, we marked these calls as non-critical by appending “|| true”, and noted in this report that this verification step is limited to cluster-internal access.

Development and Debugging Lessons

1. Refactoring the stats.py Function

The initial “stats.py” had code which created empty outputs or else generated HTTP 500 errors as an effect of invalid field accesses and missing error processing. Once the function was rewritten according to the structure and conventions outlined in the demo script—like safely parsing aggregation responses, using field names accurately, and improved error reporting—the function consistently produced expected outputs. This emphasized the importance of referencing clean, tested templates when it comes to production functions.

2. Bluesky Harvester Deployment Failure

Function initialized but not triggered The bluesky-au function was successfully implemented in Fission and the logs verified that the loading of the module was successful. The function never fired, though, with the consequence of not fetching and indexing any data.

Firstly, the main() block is incompatible with Fission. The script was depending on if __name__ == "__main__": to run the harvesting logic. Fission's event-driven structure ignores this entrypoint, so the main logic was never run. Secondly, no response observed upon invocation Despite deployment, subsequent HTTP calls to the function yielded no significant response. The lack of return statement or handler that is HTTP-compatible caused it to be hard to confirm successful execution. Thirdly, lack of visible execution effects The logs indicated success in startup but no sign of fetching or processing the posts. Lacking direct feedback from the function to diagnose the problem. Meanwhile, mixing local and cloud logic The code had both local execution and cloud deployment logic embedded in it, thereby creating confusion as to how the function must be defined or invoked in Fission.

3. Function Testing Challenges: Empty Results with Complex Queries
The function produced an empty list despite the presence of relevant documents in the Elasticsearch index. This led to ambiguity in terms of whether the Fission function's logic was broken, whether the query build was erroneous, or whether the underlying data had changed in some way or become invalid. Specifically, the problem was the presence of multiple filters—date range, keyword match, and sentiment score threshold—that were all needed to be true at the same time. If none of these was true, Elasticsearch would just not return anything and produce an empty aggregation outcome.

The issue was further complicated by the fact that the query was generating a successful 200 response with an empty body, giving the impression the function had run successfully but had not produced useful results. This required closer examination of the actual Elasticsearch query_body itself, as well as checking field mappings, filter logic, and test case sufficiency.

This case highlighted the importance of testing Elasticsearch queries independently and confirming that aggregation conditions are met before interpreting the response as meaningful output.

4. Deployment Drift and Regression Debugging
An earlier functioning script started producing no output after a series of what appeared to be minor code modifications. These ranged from variable name changes, filter alterations, to refactoring bits of the query. Even though each modification seemed to be innocuous in isolation, collectively these introduced regressions in the pipeline for execution. In other situations, the updates were not being pushed successfully to Fission, and the function being run did not contain the most current code. This introduced a disparity between what was anticipated and what was being run in the cloud.

The absence of obvious errors made the problem especially hard to detect because the function would continue to return an HTTP 200 response and empty responses, providing a false indication of success. Debugging involved rolling back to an earlier working state of stats.py, adding changes incrementally back in, and utilizing logs (kubectl logs) and successive curl requests to identify the specific modification that introduced the problem. The process emphasized the need for disciplined control of versions and meticulous deployment testing in serverless environments such as Fission.

Analytical Strategy Adjustment
Because we have previous experience on connecting Mastodon to Elasticsearch and the experience of deploying YAML in the cloud, it is easy to modify the code on Reddit's data extraction, and the cloud deployment is also very successful. However, the function continuously timed out. We thought it was a code related problem or a Fission timeout problem. Because the default timeout of Fission is 60 seconds, we set it to 120 seconds, but it still doesn't work. Then we changed the keyword to 'a', a much common word, the function ran successfully, Elasticsearch could catch lots of posts. This means there is no problem with the code, just simply Reddit chat containing our posts, so it means the code is fine. Simply meaning that Reddit did not contain many posts about our topic. Actually, this is normal because the Australian communities are not paying much attention to Trump, and tend to focus on local topics such as travel, lifestyle and food. So we have thought about changing the community we catch to "news", which is more of a community that talks about big world events, but how do we tell if it's an Australian posting.Our idea is to firstly identify users who have posted Trump-related posts, and find out which communities they have joined by searching for their usernames, and if they have joined specific Australian communities (e.g., Melbourne, Sydney, Brisbane), then we assume this user is an Australian. But after we modified the code to this, we kept having problems, and finally we reverted to the original plan, which is retrieving posts from Australian-based communities with key words "Trump" or "tariff". Luckily, we got at least 11+ posts. It is valuable to be used to compare sentiment of individuals across two platforms.

Function Deployment and Testing Workflow
1. Debugging ES Query Fission Function Creation
The "demo" function was used to connect with HTTP routes and retrieve required data from ES. At the beginning, the corresponding yaml specifications such as route, package, function yaml specifications were defined. Upon applying them in Fission, it showed the demo package remained running which indicates that an error occurred during creating the demo package. To resolve the issue, the specifications were stopped, and only created the demo package. However, the package continued to appear as "running" in the Fission package list. Use "chmod +x build.sh" to ensure execution permission for build.sh, and then review the "requirements.txt" to verify whether it includes all required functions, but the package still remained in "running". To further check the problem, a local virtual environment was created to manually run "build.sh" to identify which step pip encountered issues. After running on-premise, the function and its directory were renamed in order to avoid naming conflicts. This can ensure that Fission will create the correct package from the directory rather than using a ZIP file with the same name. Then reapplied the YAML specifications, ultimately it successfully created the correlated package and function. The demo function was tested to verify whether it can query data from Elasticsearch as intended. Then use this test command--"curl http://localhost:9090/analysis/date/2025-05-12 | jq '.'"--to confirm that the function could return expected results.

However, a new issue emerged, and it returned "TypeError: The view function did not return a valid response", which indicates that a Python list was directly returned by the Flask function in demo.py. This does not comply with Fission's expected response format (JSON). To solve this,"jsonify"is imported from Flask and is used to modify the return statement to"jsonify(result) ". This enabled the function to return an empty list [], which means all functional links were fully operational. Despite this, the function still failed to collect data from Elasticsearch. After checking ES connection and confirming that there were records available for the given date in the data view, probably the issue was related to the query format. Specifically, the date field in the query did not match the format stored in the index. Upon searching the time field format, it was clear that the format should be "created_at": "2025-05-12 15:19:44", but the original demo function "date_expr" template was set by "${date}T00:00:00". After updating the template to match the correct format ("2024-05-10 00:00:00"), the function ran successfully and retrieved the expected data from ES.

2. Testing CI/CD Pipeline
We initially used the "fission function test" command in the "test-fission-function" job, but it failed. This command depends on access to the Fission router, which is only reachable within the Kubernetes cluster network. Because GitLab CI runs externally and the router service was not exposed to the public, the request could not be routed. We tried to solve it by adding --createingress to routes, but it had no effect. Finally, we renamed this to "check-fission-functions" and replaced this execution to verify whether Fission functions are created by specs or not in local environments.

When executing the "fission spec apply" command for the second time in CI, the "deploy-fission-specs" job failed and showed the "HTTPTrigger already exists" conflict error. Because the Fission HTTP routes cannot be created repeatedly (they have been created before in the first CI pipeline). To solve this problem, we added a cleanup step in CI, including executing the "fission httptrigger delete --name <route-name> || true" to delete the old route before deployment, and also remove the unused route configuration files from the specs directory.

Git collaboration Conflicts
During the previous update, the GitLab repository was unintentionally overwritten because of the use of "git push --force" while trying to upload the "specs/" directory for Fission deployment. All original files have since been manually restored, and inquired whether other teammates have old logs for the files.

Teamwork and Contribution

Collaboration Tools and Workflow
Over the course of the project, our team collaborated well on a regular basis. Every week we meet on Zoom to review our overall progress and plan the next sprint. During those meetings the team leader distributed specific tasks such as harvesting functions, Elasticsearch queries and Jupyter notebook visualizations to team members and tracked each member's completion of their assignments. Outside of our scheduled meetings, we maintained an active WeChat group where anyone who faced a challenge could post a question and receive prompt guidance from the team leader or whichever teammate who had relevant experience. All code changes went through our shared GitLab repository on main, with each contributor opening merge requests for peer review before rebasing and merging keeping everyone informed of one another's work through the commit history. With weekly meetings, prompt problem-solving through group chat, and collaborative code reviews in place, we ensured that no one felt stuck and that all deliverables advanced smoothly toward our project completion.

Collaboration and Conflict Resolution

Throughout development, our repository experienced a steady stream of commits—primarily by Zijing Li—touching everything from backend data processing to frontend UI. To integrate these changes without creating a tangled commit graph, we adopted a strict “pull, rebase, resolve conflicts, then push” routine. Before each push, we fetched the latest main and rebased our feature work on top of it, at which point any overlapping edits would be flagged. We handled each conflict by opening the affected files, comparing both versions, merging the new logic into the existing code, and marking the conflicts resolved before continuing the rebase. Because we rebased rather than merged, our history remained linear and merge-commit–free, making code reviews simpler and keeping our CI pipelines from failing due to transient inconsistencies. This disciplined workflow allowed us to incorporate file uploads, deletions to API tweaks and scheduling adjustments smoothly into the main branch with minimal friction.

Contribution Table

Name	Tasks
Zijing Li	“Structure” + “Installation” in README.md and configured Kubernetes settings; Created and applied all timers, functions and packages specs on the cloud; Updated sentiment.py; Adapted on-premise harvest functions for cloud deployment; Developed CI pipeline; Developed the “pharvest” function; Developed the “mcontent”, “timestats”, “subredditstats” functions; Developed final-version Jupyter Notebook; Participated in “System architecture” + “past data harvesters” + “yaml specs CI pipeline” parts in Youtube Demo; Contributed to the final report and worked on all team meetings.
Henjin Hou	Worked on the old Jupyter Notebook and handled the data analysis section; Participated in “Jupyter notebook” part in Youtube Demo; Contributed to the final report and attended all team meetings.
Binzhen Wei	Developed on-premise harvest functions and conducted tests to verify their functionality; “Instructions” + “Installation” in README.md; Developed the “rstats” and “stats” functions; Developed the keyword+date range routes specs; Updated and applied keyword and day-range routes specifications on the cloud; Participated in “Harvester functions” part in Youtube Demo; Contributed to the final report and attended all team meetings.
Wenquan Wan	Tested the on-premise query function on the cloud; Participated in “ES query functions” + “Fission route” parts in Youtube Demo; Contributed to the final report and attended all team meetings.
Zeyu Chen	Developed basic sentiment.py script; Developed the basic es query code; Drafted routes specs; Participated in “Scenario” part in Youtube Demo; Contributed to the final report and missed one team meeting.

Conclusion and Future Work

In this project, we successfully constructed a cloud-based big data analysis system, which enables real-time data collection, processing and visualisation of "Trump" and "tariff" topics on two social platforms, Mastodon and Reddit. Based on Kubernetes, Fission, and Elasticsearch, we designed a scalable and modular data processing architecture, which supports streaming data capture and historical data retrospective. It combines keyword-based sentiment analysis with multiple visualisation tools to dynamically display results in the Jupyter Notebook frontend. In terms of analysis scenarios, we focus on exploring how the Australian public perceives Trump and his tariff policy. Through sentiment trend graphs, keyword evaluation and cross-platform comparisons, we found that discussions related to "Trump" elicited stronger engagement and emotional fluctuations than "tariffs". The overall sentiment of Mastodon users was negative, especially after the announcement of the tariffs. In contrast, sentiment on Reddit was more fragmented and regionally varied. In addition to achieving technical goals, our team adopted good engineering practices in version control, task allocation, and collaborative development. GitLab was used for code management and continuous integration to ensure efficient collaboration and smooth operation of all modules.

Although our project demonstrates strong completeness and practicability, there is still room for improvement in the following aspects:

1. Expanding data coverage: Due to deployment problems , Bluesky platform wasn't included in the analysis process. In the future, resolving network configuration problems or optimizing the function structure could enable successful integration.
2. Enhancing geographic identification: The identification of "Australian users" in Reddit analysis still remains relatively rough as the reddit related posts in the subreddits we chose are not enough. In the future, if we have more reddit posts data, more accurate geographical filtering can be achieved by incorporating user information, timestamps, and community activity records.
3. Upgrading sentiment analysis models: Current system relies on the VADER Sentiment lexicon, which has limitations in detecting complex terms such as sarcasm and double meanings. BERT or RoBERTa could be introduced in future to improve the accuracy of analysis.
4. Dynamic analysis of user behaviour: Currently, the analysis focuses on keywords usage and sentiment distribution, but in the future, it can be further mined to find out user behavioural patterns such as multi-posting users or topic migration trends.
5. Enhancing interactivity and visualization: Although the current Jupyter Notebook provides basic display functions, its interactive capabilities are limited. It can be added, such as seconds/minutes posts count graphs, the same as the dataview in Kibana, which can directly show the data stream.
6. Past data function: The “pharevst.yaml” job can be improved by querying the Elasticsearch index for the oldest stored post and using it as the max_id automatically. Then it can be deployed as a Fission function interacting with the timer to accomplish automation.

Reference

[1] Australia Times. (2025, May 20). *Impact of Trump's tariffs on the Australian economy raises concerns*. https://australiatimes.com/impact-of-trump-s-tariffs-on-australian-economy-raises-concerns
[2] Grantham-Philips, W. (2025, April 3). *A timeline of Trump's tariff actions so far*. *PBS News*. https://www.pbs.org/newshour/economy/a-timeline-of-trumps-tariff-actions-so-far
[3] Martin, J. (2025, April 6). *Will Trump's sweeping global tariffs derail Australia's economy? Chalmers offers reassurance*. *7NEWS*. https://7news.com.au/news/us-tariffs-expected-to-have-modest-impact-on-australian-economy-according-to-treasury-analysis-c-18291520
[4] Hutto, C. J., & Gilbert, E. (2014). VADER: A parsimonious rule-based model for sentiment analysis of social media text. In *Proceedings of the Eighth International AAAI Conference on Weblogs and Social Media (ICWSM 2014)* (pp. 216–225). AAAI Press.
[5] KEDA. (n.d.). *KEDA – Kubernetes-based event-driven autoscaling*. Retrieved May 20, 2025, from https://keda.sh/