COMP90024 TEAM 54: Cluster and Cloud Computing

ASSIGNMENT 2

Topics: Social Media Analytics for AFL and Urban Transport Using Cloud Technologies

NAME	STUDENT ID
Xueying Yuan	1531588
Shuangquan Zheng	1331085
JunJie Wang	1103084
Hao Xu	1468277
Chunmiao Zheng	1642700

Contents

1 Introduction

This project builds a scalable analytics platform on the MRC/NeCTAR Research Cloud for processing large volumes of social media data. It collects posts and comments about AFL and public transportation from Reddit, Mastodon, and Bluesky. This project needs to use Kubernetes to run and manage all services. It also uses Fission to create serverless functions that collect and clean data when needed. In addition, this project stores and index data in Elasticsearch for analysing and create visualization.

Our team goals are to develop a range of analytic scenarios comparing harvested/streamed external data. Our team's scenarios are about AFL games and city transport and extract related posts and comments from Reddit, Mastodon, and Bluesky. Our team also scores sentiment for each post and tracks how many fans follow each team and people view on public transportation in different cities. Moreover, we compare sentiment scores and look at differences between platforms.

2 Discussion: Pros and Cons of the NeCTAR Research Cloud and Tools

The National eResearch Collaboration Tools and Resources (NeCTAR) Research Cloud, provided by the Australian Research Data Commons (ARDC), offers a cloud-based infrastructure specifically designed to support research projects and allow us to. It has established data storage resource across Australia, it has multiple availability zones in contrast to (Melbourne Research Cloud) MRC which only has a single data centre with limited number of IP address, no GPU and only one availability zone. NeCTAR allows its users to deploy and manage virtual machines, host services like databases and web portals, and carry out high volume data analysis. The platform is self-service and supports collaboration across institutions, useful for nationally projects that requires collaboration with users external to the university giving it's an advantage over MRC. Which is why NeCtar was chosen for this project to deploy a Kubernetes cluster and host various services such as ElasticSearch and Fission for data harvesting and sentiment analysis.

2.1 National eResearch Collaboration Tools and Resources (NeCTAR)

NeCTAR offers a cloud-based infrastructure tailored for research projects enables it user to deploy and manage virtual machines, host services such as databases and web portals, and perform large-scale data analysis through a user-friendly, self-service platform. It also supports cross-institutional collaboration, making it especially valuable for national research initiatives that involve users external to university.

For this project, NeCTAR was selected over MRC to deploy a Kubernetes cluster and host essential services such as ElasticSearch, Fission, KEDA, and Redis. These tools work together to allow us to automate data harvest functions, easy scaling in the future and event driven workload orchestration. NeCTAR's flexibility and robust infrastructure provided the foundation for building a resilient, scalable, and collaborative research platform.

2.2 Resource Allocation

We deployed a Kubernetes-based architecture on the NeCTAR Research Cloud, using multiple virtual machines (VMs) to support crucial processes such as distributed data ingestion, real-time sentiment analysis, and serverless function execution. A total of four nodes were provisioned using the "uom.mse.2c9g" flavor, each providing 2 vCPUs and 9 GB of RAM per instance.

Collectively, our deployment utilised 8 vCPUs, 36 GB of RAM, and 209 GB of volume storage, as reflected in the limit summary below. These resources were essential in running containerized services such as ElasticSearch, Redis, and Fission.

```
openstack coe cluster create\
  --cluster-template "kubernetes-v1.31.1-melbourne-qh2-uom-v4" \
  --node-count 3 \
  --master-count 1 \
  --master-flavor "uom.mse.2c9g"\
  --flavor "uom.mse.2c9g"\
  comp90024
```

Figure 1:

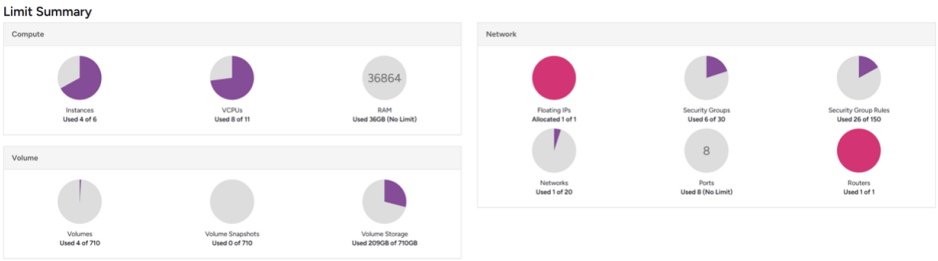


Figure 2:

2.3 NeCTAR Research Cloud Advantages

One of the major benefit others than its free access for students is its ability support self-service virtual machines (VMs) due to its IaaS nature, enabling our team to quickly provision and configure instances for deploying a Kubernetes cluster. This self-service feature allows us to have full control over the configuration of the compute instance, volume and networks. This the model cost effective, as we can control the resources we use and scale or reduce the resource depending on the demand, which helps minimise waste and avoid overprovisioning.

Additionally, NeCTAR integrated with OpenStack tools allows flexible resource scaling and network configuration. The detailed dashboards NeCTAR has available shows quotas and usage statistics for instances, volumes, IPs, and ports further assisted in managing resource allocations effectively.

2.4 NeCTAR Research Cloud Disadvantages

Despite its advantages, NeCTAR does present several challenges. Security being a primary concern, particularly for projects involving sensitive or regulated data. Human

error is common risk when building application and the public cloud magnify this risk, as infrastructure is shared among multiple tenants, there is an increased risk of data breaches or unauthorised access. Furthermore, users often experience a loss of control over the underlying infrastructure, limiting their ability to customise configurations or address hardware-level issues, which did occur for some users and delay their work due to their inability to resolve issue them as they have relied on the IT. Another consideration is the potential for lock-in, where dependency on its cloud services or APIs can make it difficult for us to migrate to alternative platforms such as Azure in the future. Finally, we heavily on relies on the long-term viability and maintenance of the service provider, if any changes happened to NeCTAR for example loss of funding, then it could disrupt ongoing research workflow.

3 System architecture and Dynamic Scaling

Delivery of the platform relies on integration of several open-source tools that supported container orchestration, serverless execution, scalable function triggering, and real-time data storage. Key technologies we employ in project includes Kubernetes for container orchestration, Fission for serverless function management, KEDA for event-driven autoscaling, Redis for queue management, and ElasticSearch for indexing and querying sentiment data.

3.1 Kubernetes

Kubernetes was deployed on the NeCTAR Research Cloud to manage containerised services across multiple virtual machines as it is recognised as the industry standard for container orchestration. It provided a flexible foundation for deploying software, managing pods, and scaling components dynamically to ensure service continuity. Kubernetes achieves this through its ability to distribute the container workload across a cluster of nodes in this project, consisting of one master node and three worker nodes. It also supports the distribution of configuration data to containers, allows the definition of environment variables within containers through config maps, and enables the use of namespaces to host multiple versions of the same application in isolation.

However, Kubernetes also presented several challenges. Resource quotas imposed by NeCTAR limited horizontal scalability and led to pod scheduling issues. Additionally, 'CrashLoopBackOff' and 'Evicted' state frequently occurred (Fig. 3) due to resource exhaustion or misconfigured containers. This errors often prevent our group from building packages. Debugging these issues required manual inspection of pod logs and resource metrics, increasing the complexity to the deployment process. Furthermore, 'Evicted' pods do not restart automatically and must be manually deleted, increasing administrative overhead and reducing the efficiency of the deployment workflow.

```
NAME                                READY   STATUS                   RESTARTS   AGE
buildermgr-6cf47b9dd9-r2tvh         1/1     Running                  0          9d
executor-54856f45c-hkhj2            1/1     Running                  0          9d
kubewatcher-5849d656df-vtlg4        1/1     Running                  0          9d
mqtrigger-keda-685884c48f-r9bqz     1/1     Running                  0          9d
router-5c7975fb76-pjsbq             1/1     Running                  0          9d
storagesvc-77cc5697b9-22cm9         0/1     Evicted                  0          23h
storagesvc-77cc5697b9-2n2ht         0/1     Evicted                  0          23h
storagesvc-77cc5697b9-2z8pq         0/1     ContainerStatusUnknown   1          6d12h
storagesvc-77cc5697b9-4pdr5         0/1     Evicted                  0          23h
storagesvc-77cc5697b9-5712p         1/1     Running                  0          23h
storagesvc-77cc5697b9-5t4xb         0/1     Evicted                  0          23h
storagesvc-77cc5697b9-5vz8v         0/1     ContainerStatusUnknown   1          9d
storagesvc-77cc5697b9-7rvch         0/1     Evicted                  0          23h
storagesvc-77cc5697b9-b97qm         0/1     Evicted                  0          23h
storagesvc-77cc5697b9-bcd9j         0/1     Evicted                  0          23h
storagesvc-77cc5697b9-fgt22         0/1     Evicted                  0          23h
storagesvc-77cc5697b9-fqqlb         0/1     ContainerStatusUnknown   1          2d4h
storagesvc-77cc5697b9-g8knm         0/1     Evicted                  0          23h
storagesvc-77cc5697b9-gd6zw         0/1     Evicted                  0          23h
storagesvc-77cc5697b9-kqr96         0/1     Evicted                  0          23h
storagesvc-77cc5697b9-lbf96         0/1     Evicted                  0          23h
storagesvc-77cc5697b9-lvxmb         0/1     Evicted                  0          23h
storagesvc-77cc5697b9-mj85h         0/1     Evicted                  0          23h
storagesvc-77cc5697b9-mkpmw         0/1     Evicted                  0          23h
storagesvc-77cc5697b9-n5wfs         0/1     Evicted                  0          23h
storagesvc-77cc5697b9-pzcvx         0/1     ContainerStatusUnknown   1          3d21h
storagesvc-77cc5697b9-ssss6         0/1     Evicted                  0          23h
storagesvc-77cc5697b9-tzvmx         0/1     Evicted                  0          23h
storagesvc-77cc5697b9-wngbk         0/1     Evicted                  0          23h
storagesvc-77cc5697b9-xxjrk         0/1     Evicted                  0          23h
storagesvc-78f84f7c4d-mx4bj         0/1     ContainerStatusUnknown   1          17d
timer-866c684989-c5lzs              1/1     Running                  0          38h
webhook-7c86b98f-j9gcn              1/1     Running                  0          17d
```

Figure 3:

3.2 Fission

Fission was employed as the framework for serverless functions on Kubernetes, allowing users to deploy functions such as sentiment analysis, data cleaning and storing to run independently and on demand. While Fission allows quick and simple deployment of lightweight functions, it also introduced difficulties during the build and runtime phases. Functions occasionally failed to build due to package version mismatches or improperly formatted archives. To optimise execution latency, the PoolManager executor was used however it struggles to handle heavy load on a single function hence some of our harvester function timeout due to this limitation. It's alternative NewDeploy executor can process heavy work however this resulted in higher latency which can be avoided if one pod is active at all times which could result in redundant resource usage which is not ideal in environments like NeCTAR when quotas are limited in this project.

3.3 ElasticSearch

ElasticSearch a document-oriented DBMS employed as the core storage and analytics engine in our architecture, enabling real-time querying data harvested for from social medias via API. As shown in Fig. 4, our system used a serverless architecture where Fission functions harvested social media data and forwarded cleaned documents to ElasticSearch via REST API calls.

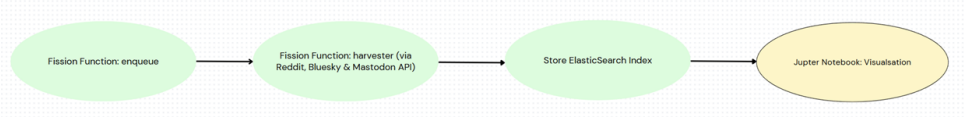


Figure 4:

ElasticSearch was chosen for its distributed nature and ease of deployment within Kubernetes. Its HTTP based API made interaction straightforward for out fission functions, and its powerful GUI, Kibana, allowed us to inspect and validate indexed data. Additionally, ElasticSearch benefits also include full-text search, support for time-based data retrieval, and a flexible querying model, making it well suited for the unstructured data harvested from social media platforms.

ElasticSearch also supports sharding and replication, which facilitates horizontal scaling in our Kubernetes-based NeCTAR deployment. Sharding helped distribute computational load across nodes, while replication improved fault tolerance. Together, these features maximised availability while maintaining a level of data safety.

Despite its strengths, ElasticSearch presented some limitations in our project. It has difficultly to join data to relate posts and comments across different social media sources. In this project we stored AFL-related data in separate indexes (e.g., Reddit, Bluesky), but cross-index aggregation was inefficient and required workarounds due to the absence of relational capabilities.

ElasticSearch also lacked multi-document transaction support, increasing the risk of partial updates when handling batches of documents. Frequent updates led to index size growth, as ck function to avoid duplicate inserts and reduce unnecessary operations.

3.4 Redis and KEDA

To enhance the system's scalability, Redis and KEDA were introduced into the architecture. Redis acted as a lightweight, in-memory message queue, enabling asynchronous communication between ingestion (enqueue see Fig. 7) and processing (harvester see Fig. 6) stages. This separation allowed the system to buffer tasks during traffic spikes and prevent function overload to avoid timeouts that previously occurred when invoking the harvester Fission function (see Fig. 5).

```
Usage:
  fission function test [options]

Error: error executing HTTP request: Get "http://127.0.0.1:51162/fission-function/hello-1": function request timeout (60000000000)s
 exceeded
```

Figure 5:

```
fission mqtrigger create --name afl-harvesting \
--spec \
--function aflharvester \
--mqtype redis \
--mqtkind keda \
--topic afl \
--errortopic errors \
--maxretries 3 \
--metadata address=redis-headless.redis.svc.cluster.local:6379 \
--metadata listLength=100 \
--metadata listName=afl:subreddit
```

Figure 6:

```
fission httptrigger create --spec --name enqueue --url "/enqueue/{topic}" --method POST --function enqueue
```

Figure 7:

To make the platform more adaptive to load fluctuations, KEDA (Kubernetes Event-Driven Autoscaling) was integrated to monitor Redis queue and automatically scale Fission worker functions up or down. This event-driven model ensured efficient resource usage by allocating compute power only when needed, which was particularly important within the quota-constrained NeCTAR environment. However, both Redis and KEDA introduced their own challenges. Redis required careful memory management and is limited in data storage, posing a risk of data loss. KEDA, on the other hand encounter the issue of cold start latency when scaling from zero which can introduce latency for functions that requires immediate response. These additions made our serverless pipeline significantly more robust and adaptive to real-time data surges.

3.5 Dynamic Scaling

Our serverless architecture was designed for the purpose of dynamic scaling, allowing us to integrate new social media harvesters with minimal disruption. Each harvester in our cluster was implemented as an independent Fission function, responsible for fetching and processing data from a specific platform (e.g., Reddit, Bluesky and Mastodon). Adding a new harvester for instance, to support Twitter or Facebook only required deploying a new function and registering it with a message queue via Redis.

The pipeline was supported by prebuilt functions such as addElastic for storing data in ElasticSearch, checkElastic for duplicate detection, textClean for noise removal, and enqueue, which routes platform-specific data into the Redis queue based on a topic parameter (see Fig. 8). We also employed a shared ConfigMap named shared-data to centrally manage common resources, such as the list of monitored subreddits (see Fig. 9), promoting consistency and ease of updates. Sensitive credentials were stored using Kubernetes Secrets, allowing all team members secure and unified access to sensitive credentials such as platform tokens and API keys. As each function focused on a single responsibility, new data sources could be rapidly integrated into the pipeline while relying on shared functions for storage, deduplication, and preprocessing. By making use of these SaaS-style Fission functions, we were able to achieve dynamic scaling, enabling new data sources to be integrated seamlessly while maintaining consistent processing across platforms.

Thanks to our use of KEDA, which monitors Redis queue depth, the system could automatically scale the corresponding harvester pods up or down based on incoming data volume. This modular design ensured that new data sources could be integrated quickly without affecting existing services, demonstrating the flexibility and scalability of our architecture. Furthermore, the Redis-backed queue decoupled ingestion from processing, allowing each new harvester to operate independently while still contributing to the central ElasticSearch index for downstream analysis.

```
addelastic      python39    poolmgr    0    0    0    0    0    0    elastic-secret                default
checkelastic    python39    poolmgr    0    0    0    0    0    0    elastic-secret                default
enqueue         python39    poolmgr    0    0    0    0    0    0    elastic-secret    shared-data default
textclean       python39    poolmgr    0    0    0    0    0    0                                   default
```

Figure 8:

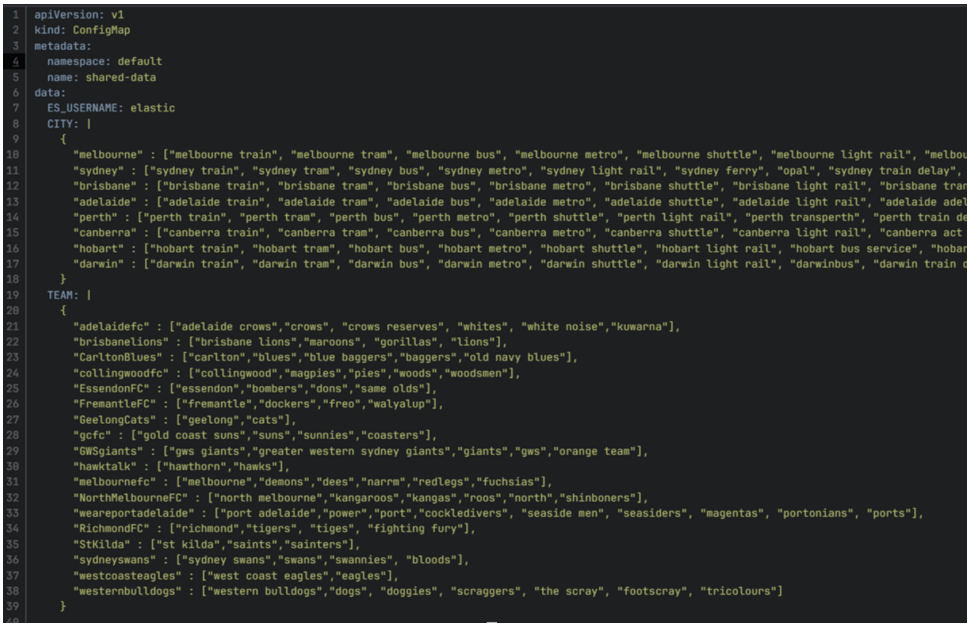


Figure 9:

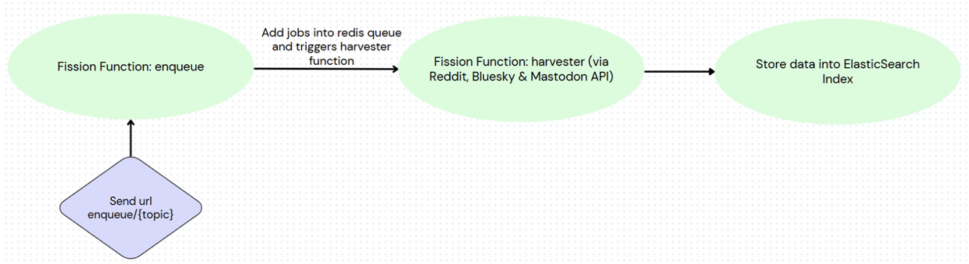


Figure 10:

4 System Modules and Functionality

4.1 Data Harvesters

4.1.1 Reddit

Data harvesters pull content from Reddit and use the `PRAW` library:

- `Harvester.py`: Invoke the above mentioned fission functions and use VADER sentiment analysis and call Reddit.
- `SubRed.py`: Fetches transportation Reddit posts filtered by subreddit.

Authentication:

- Use Reddit's OAuth2-based authentication with `client_id`, `client_secret`, and `user_agent`.

4.1.2 BlueSky

Data harvesters pull content from Bluesky and use the `httpx` library:

- `harvest_aflbluesky.py`: Authenticates with Bluesky, performs keyword-based search for AFL-related posts using team nicknames, applies VADER sentiment analysis, and stores sentiment-tagged posts in ElasticSearch if not already indexed.
- `blueskyfan.py`: Fetches followers of official AFL team accounts on Bluesky and stores structured follower metadata (e.g., handle, display name, follower count) for network analysis and fan base tracking.

Authentication:

- Uses session-based login via Bluesky's `com.atproto.server.createSession` endpoint.
- Credentials (username and password) are securely read from Kubernetes-mounted secrets and passed to obtain a bearer token for API access.

4.1.3 Mastodon

Data harvester pulls content from Mastodon and uses the `Mastodon.py` library:

- `mharvester.py`: Fetches public transport-related posts on Mastodon by querying city and topic-related keywords (e.g., "melbourne train", "sydney tram") from the `mastodon.social` instance. Cities include Melbourne, Sydney, Brisbane, Adelaide, Perth, Canberra, Hobart, and Darwin.

Authentication:

- Use Mastodon's token-based authentication with `mastodon_token`. The token is mounted as a Kubernetes secret and securely read by the Fission function during runtime.

Filtering Logic:

- Due to the lack of subreddit-style structure on Mastodon, a keyword-based filtering mechanism is implemented. Posts are selected only if they contain at least one strong keyword (e.g., "tram", "myki", "vline"), and the sum of strong and weak keywords (e.g., "delay", "crowded") is ≥ 2. Posts with banned keywords (e.g., "ranked", "run", "weekend") are excluded to avoid false positives.

4.2 Data Processing

A Fission function cleans text:

- Remove emojis
- Remove special characters like double quotes, colons, newlines, non-breaking spaces, and special punctuation.
- Return normalized JSON with the cleaned text.

This ensures that the data is in a consistent and easier format before analysis.

4.3 Sentiment Analysis

A Fission function runs VADER sentiment analysis. It scores positive, neutral, and negative tones:

- Collect sentiment polarity (positive, neutral, negative) and produces a compound score for each text.
- Use in the Harvester functions. Get total weighted sentiment per city, if city is not mentioned in posts, then assumed it's related to subreddit city.
- Results are tagged with metadata like city, score and type and sent to ElasticSearch.

4.4 Fission Functions

Several Fission functions manage data input, output and conversion:

- `addElastic`: Initialise ElasticSearch and add documents based on index variable given to function.
- `checkelastic`: Verify if a document with a specific `docID` exists. If exist, skip.
- `enqueue`: Add new jobs to Redis queue based on current data volume and topic param given.
- `textclean`: Clean Reddit post text to remove unnecessary characters.
- For small data harvest function, we set timer to run every 5 minutes and for large data harvest we trigger enqueue function by calling fission function set with 5 minutes timer in turns invoke harvester function
- `Secrets and ConfigMaps`: Holds

Trigger Mechanism:
Functions use Redis MQ triggers. KEDA scales pods based on queue length. The `mqtrigger` command binds functions to Redis lists. For example:

```
fission mqtrigger create --name trans-harvester-mqtrigger \
    --spec \
    --function transharvester-reddit \
    --mqtype redis \
    --mqtkind keda \
    --topic trans:subreddit \
    --resptopic observations \
    --errortopic errors \
    --maxretries 3 \
    --metadata address=redis-headless.redis.svc.cluster.local:6379 \
    --metadata listLength=100 \
    --metadata listName=trans:subreddit
```

Figure 11:

It also specifies topics, retry policies, and queue details.

4.5 Social Media Data Pipeline: Harvesting, Processing, and Storage Functions

Multiple harvester functions have been designed to collect and process data for the scenarios we chose for this assignment. Each harvester is tailored to meet specific requirements based on the data source and the type of analysis to be performed. Below is an overview of each of these harvesters and how they operate.

4.5.1 Reddit – AFL: Introduce to `aflHarvest.py`

- **Function trigger by Redis Queue**

  This consumer function is designed to be triggered when jobs are added to a Redis queue. Each job contains information about the subreddit to harvest and the post limit for processing. The function uses Redis to manage the queue. The job is pulled from the queue (`rpop`), which includes details such as the Reddit subreddit and the number of posts to process. After retrieving the job from the producer function enqueue, the consumber function aflHarvester proceeds to fetch post from specified subreddit.

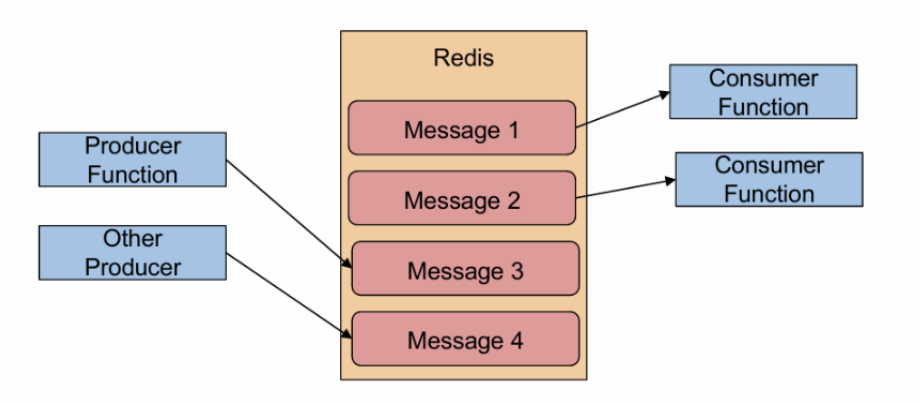


Figure 12:

- **Reddit API**

  The function uses PRAW (Python Reddit API Wrapper) to interact with Reddit and fetch posts from specific subreddits using credentials stored in secrets.

```python
def main():
    """
    Run harvest subreddit for all AFL team, skip if we get an conflicterror
    Return: return string and 200 code after harvested afl data
    """
    current_app.logger.info(f'=== aflHarvester: Initialise ===')
    redisClient: redis.StrictRedis = redis.StrictRedis(
        host='redis-headless.redis.svc.cluster.local',
        socket_connect_timeout=5,
        decode_responses=False
    )

    try:
        team = redisClient.rpop("afl:subreddit")
        if not team:
            return "Queue empty", 200
        job = json.loads(team)
        harvestSubreddit(job["team"], postLimits=job["limit"])
        current_app.logger.info(f'=== aflHarvester: Job completed Harvested {job["team"]} ===')
        return f"Harvested {job['team']}", 200
    except ApiError as e:
        return json.dumps({"error": str(e)}), 500
```

Figure 13:

```
with open("/secrets/default/elastic-secret/REDDIT_CLIENT_ID") as f:
    clientID = f.read().strip()

with open("/secrets/default/elastic-secret/REDDIT_CLIENT_SECRET") as f:
    clientSecret = f.read().strip()
reddit = praw.Reddit(
    client_id=clientID,
    client_secret=clientSecret,
    user_agent="python:reddit-harvester:v1.0 (by /u/Exact_Agency_3144)",
)
subreddit = reddit.subreddit(redditTeam)
current_app.logger.info(f'=== aflHarvester: Process reddit post/comments ===')
for post in subreddit.new(limit=postLimits):
    print(post.url)
    text = cleanText(post.title + " " + post.selftext)
    postTeams = teamMentioned(text)
```

Figure 14:

- **Text Cleaning**

  Before sentiment analysis is performed, the text content is cleaned using the **Fission** function `text-clean`. This step helps to remove unwanted characters, special symbols, and other noise in the data that may affect VADER sentiment analysis.

```
def cleanText(text)  -> str:
    """
    Send text to fission fucntion text-clean for cleaning and return cleaned text
    """
    url='http://router.fission/text-clean'
    payload = {"text":text}
    response = requests.post(url,json=payload)
    return response.json()["cleanedText"]
```

Figure 15:

- **Team mentioned**

  The function detects whether the post or comment mentions any AFL team using a predefined dictionary `TEAM`. This dictionary maps AFL team names to common nicknames. The `teamMentioned()` function checks the text to see if any of the team nicknames appear, returning a list of identified teams.

- **Sentiment Analysis**

  VADER `SentimentIntensityAnalyzer` function was used to perform sentiment analysis on the text. The sentiment is weighted based on the post's upvote score and number of times teams were mentioned in the text, using the `sentimentPerTeam()` function. The sentiment score is computed for each team mentioned in the post or comment, and if no team is mentioned, then the sentiment is assigned to the subreddit (team).

- **Storing Data in ElasticSearch**

  The processed sentiment data is assigned into the doc schema and then stored into ElasticSearch using the fission function `addElastic()`. It checks if a post or

comment is already indexed using fission function `checkPost()`, if then it will be skipped to avoid duplication. Each document is indexed under the `afl-sentiments` index, and data such as sentiment score, text, upvote score, and post URL are stored for visualisation.

```
if checkPost(docId,"afl-sentiments"):
    current_app.logger.info(
        f'Skipped {docId} as it exists '
    )
    return "exist"
```

Figure 16:

```
doc = {"type": postType,
       "platform":"Reddit",
      "team": team.lower(),
      "sentiment":sentiment,
      "text": text,
      "upvote":post.score,
      "createdOn": datetime.fromtimestamp(post.created_utc).isoformat(),
      "url":post.url,
}
addElastic(docId,"afl-sentiments",doc)
current_app.logger.info(
    f'Stored {docId} successful'
)
```

Figure 17:

For each post, the harvester also processes its comments. The function ignores comments without team mentions, and the result will be stored in ElasticSearch similarly to the post.

- **Limitation**

  a) **API limit**
  Unfortunately, due to the restriction of Reddit API, we are restricted to 1000 posts per request, limiting our harvest ability.

  b) **Sentiment analysis accuracy**
  The accuracy of sentiment analysis is dependent on the quality of the input text. Text cleaning helps, however, nuances such as sarcasm or slang may not always be handled correctly by the sentiment analysis model. More advanced models could improve results for more complex datasets.

4.5.2 Bluesky – AFL: `harvest-aflbluesky.py`

- **Text Cleaning**

Sends the input text to a Fission microservice at `/text-clean`, which removes unwanted characters or formatting (e.g. emojis, symbols). It returns the cleaned-up version of the text.

```
# Clean text via Fission service
def cleanText(text) -> str:
    url = 'http://router.fission/text-clean'
    payload = {"text": text}
    response = requests.post(url, json=payload)
    return response.json()["cleanedText"]
```

Figure 18:

- **Team Detection**

  Looks at the given text and checks if any known AFL team nicknames appear in it. If found, it returns a list of matching team IDs.

```
# Identify which teams are mentioned in the text
def teamMentioned(text, teams=teamNickname) -> list:
    foundTeam = set()
    for nickname, team in teams.items():
        if nickname in text.lower():
            foundTeam.add(team)
    return list(foundTeam)
```

Figure 19:

- **Sentiment Analysis**

  Breaks the input text into individual sentences. For each sentence, it checks which teams are mentioned, runs sentiment analysis (positive or negative feeling), and stores the sentiment score per team. It returns a dictionary with team IDs and average sentiment scores.

```python
# Perform sentiment analysis per team based on sentence-level matches
def sentimentPerTeam(text):
    sentences = sent_tokenize(text)
    teamSentiment = {team: [] for team in TEAM}

    for sentence in sentences:
        sentiment = sentimentAnalyser.polarity_scores(sentence)['compound']
        for team, nicknames in TEAM.items():
            if any(nick in sentence.lower() for nick in nicknames):
                teamSentiment[team].append(sentiment)

    result = {}
    for team, values in teamSentiment.items():
        if values:
            result[team] = round(sum(values) / len(values), 3)
    return result
```

Figure 20:

- **Duplicate Check**

  Verifies via Fission API whether the post with this doc ID already exists in Elasticsearch to prevent duplicates.

```python
def checkPost(docID):
    url = 'http://router.fission/checkelastic'
    payload = {"indexDocument": "afl_bluesky_sentiment-18", "docID": docID}
    try:
        res = requests.post(url, json=payload, timeout=5)
        return res.json().get("found", False)
    except Exception as e:
        current_app.logger.info(f"aflBluesky: checkPost error: {e}")
        return False
```

Figure 21:

- **Elasticsearch Insertion**

  Sends post data to Elasticsearch through a Fission endpoint to store it for analysis.

```python
def addElastic(docID, indexText, doc):
    payload = {"indexDocument": indexText, "docID": docID, "doc": doc}
    current_app.logger.info(f'=== aflBluesky: AddElastic : Payload: {json.dumps(payload)} ===')
    url = 'http://router.fission/addelastic'
    try:
        response = requests.post(url, json=payload, timeout=5)
        current_app.logger.info(f'=== aflBluesky: AddElastic : Response: {response.status_code} {response.text} ===')
    except Exception as e:
        current_app.logger.info(f'=== aflBluesky: AddElastic : Exception in addElastic POST: {str(e)} ===')
```

Figure 22:

- **URL Conversion**

  This function takes a Bluesky post URI and converts it into a standard web URL format that points directly to the post on the Bluesky website. It parses the internal URI structure and reconstructs it as a publicly accessible link.

```python
# Convert Bluesky URI to public URL
def convertUriToUrl(uri: str) -> str:
    if uri.startswith("at://"):
        try:
            user, postid = uri.replace("at://", "").split("/app.bsky.feed.post/")
            return f"https://bsky.app/profile/{user}/post/{postid}"
        except:
            return uri
    return uri
```

Figure 23:

- **Content Harvesting**

  This function searches Bluesky posts using a specific keyword, then processes each post by cleaning the text, identifying mentioned AFL teams, analysing sentiment per team, and storing the results in Elasticsearch if they haven't been saved already.

```python
# Search and process Bluesky posts for a given keyword
def harvestByKeyword(keyword, headers):
    try:
        resp = httpx.get(
            "https://bsky.social/xrpc/app.bsky.feed.searchPosts",
            params={"q": keyword, "limit": 100},
            headers=headers,
            timeout=10
        )
        resp.raise_for_status()
        feed = resp.json()

        for post in feed.get("posts", []):
            record = post.get("record", {})
            text = cleanText(record.get("text", ""))
            created = record.get("createdAt", datetime.now().isoformat())
            uri = post.get("uri", "")
            url = convertUriToUrl(uri)
            upvote = post.get("likeCount", 1)
            if not isinstance(upvote, int) or upvote == 0:
                upvote = 1

            teams = teamMentioned(text)
            if not teams:
                continue  # Skip if no team mentioned

            sentiments = sentimentPerTeam(text)
            for team in teams:
                if team in sentiments:
                    doc_id = f"{uri}_{team}".lower()
                    if checkPost(doc_id):
                        current_app.logger.info(f"Skipped {doc_id} as it exists")
                        continue
                    doc = {
                        "type": "post",
                        "platform": "Bluesky",
                        "team": team,
                        "sentiment": sentiments[team],
                        "text": text,
                        "upvote": upvote,
                        "createdOn": created,
                        "url": url
                    }
                    addElastic(doc_id, "afl_bluesky_sentiment-18", doc)

    except Exception as e:
        current_app.logger.info(f"aflBluesky: Error in harvestByKeyword for '{keyword}': {e}")
        return "error"
```

Figure 24:

- **Pipeline Orchestration**

This function runs the overall harvesting process. It logs into Bluesky with credentials, retrieves an access token, and loops through all AFL team nicknames to search for matching posts. Each result is passed through the full processing pipeline to extract and store sentiment-tagged data.

```python
# Main function to run the whole harvesting process
def main():
    current_app.logger.info(f'=== aflBluesky: Initialise ===')

    # Read Bluesky credentials
    with open("/secrets/default/elastic-secret/BLUESKY_CLIENT_ID") as f:
        username = f.read().strip()
    with open("/secrets/default/elastic-secret/BLUESKY_CLIENT_PASSWORD") as f:
        password = f.read().strip()

    try:
        # Login to Bluesky to get access token
        login_resp = httpx.post(
            "https://bsky.social/xrpc/com.atproto.server.createSession",
            json={"identifier": username, "password": password},
            timeout=10
        )
        login_resp.raise_for_status()
        access_jwt = login_resp.json()["accessJwt"]
    except Exception as e:
        current_app.logger.info(f"aflBluesky: Bluesky login failed: {e}")
        return "error"

    headers = {"Authorization": f"Bearer {access_jwt}"}

    # Iterate over all nicknames to harvest matching posts
    for team, nicknames in TEAM.items():
        for keyword in nicknames:
            harvestByKeyword(keyword, headers=headers)

    return "ok"
```

Figure 25:

4.5.3 Reddit – Transportation: `transHarvester.py`

Function `transHarvester` can harvest data by using PRAW (the Python Reddit API wrapper). The function fetches new posts and comments for a given city subreddit—for example, `CITY/Melbourne` or `CITY/Sydney`.

- **Clean Text**

  Each post title and body is sent to a text-clean function to remove noise, unwanted characters and emoji.

```
def cleanText(text) -> str:
    """
    Send text to fission fucntion text-clean for cleaning and return cleaned text
    """
    url='http://router.fission/text-clean'
    payload = {"text":text}
    response = requests.post(url,json=payload)
    return response.json()["cleanedText"]
```

Fig

Figure 26:

- **City Contain**

  It converts text to lowercase and checks each alias for presence. It also collects unique city names and nicknames and avoids duplicates.

```
def cityContain(text, cities = cityNickname) -> list:
    """
    Check if any city name is in the text.
    """
    foundCity = set()
    for nickname, city in cities.items():
        if nickname in text.lower():
            foundCity.add(city)
    return list(foundCity)
```

Figure 27:

- **Computing Sentiment Score**

  Each sentence receives a compound sentiment score from VADER, and this score ranges from -1 to 1. The score is then multiplied by the sentence's upvote count, and a higher absolute value of this score indicates stronger sentiment.

```
for sentence in sentences:
    sentiment = sentimentAnalyser.polarity_scores(sentence)['compound']
    for city, nicknames in CITY.items():
        cityFound = any(nickname in sentence.lower() for nickname in nicknames)
        if cityFound:
            citySentiment[city].append(sentiment * abs(upvoteScore)) # multiple by upvote score to get weighted sentiment
    if not cityFound:
        if postSub not in citySentiment:
            citySentiment[postSub] = []
        sentiment = sentimentAnalyser.polarity_scores(sentence)['compound']
        citySentiment[postSub].append(sentiment * abs(upvoteScore))
resultSentiment = {}
```

Figure 28:

- **Harvest Subreddit**

Harvests posts and their comments to get sentiment values. It fetches recent posts and cleans each one. It also saves the latest post ID it saw. In addition, it skips posts that lack any city keywords.

- **Store Elastic**

  It puts processed sentiment data into the document and then stores the data in Elasticsearch using the function `addElastic`. It uses the function `checkPost` to check if a post or comment is already indexed. If it is, it skips this post or comment.

```
if checkPost(docId,"trans-testing"):
    current_app.logger.info(
        f'Processing {docId} successful'
    )
doc = {"type": postType,
       "platform":"Reddit",
      "city": city.lower(),
      "sentiment":sentiment,
      "text": text,
      "upvote":post.score,
      "createdOn": datetime.fromtimestamp(post.created_utc).isoformat(),
      "url":post.url,
}
addElastic(docId,"trans-reddit-sentiment",doc)
current_app.logger.info(
    f'Stored {docId} successful'
)
```

Figure 29:

4.5.4 Mastodon – Transportation: `mastodon-harvester.py`

- **Function triggered by time cron on Fission**

  This function is deployed on the Fission serverless platform and triggered every 5 minutes by a time-based cron job. The cron trigger is defined in `mastodon-cron.yaml` and is used to automatically crawl Mastodon posts related to transportation in major cities in Australia on a regular basis.

- **Mastodon API**

  The function calls Mastodon's `/api/v2/search` endpoint with a predefined list of city + topic keyword combinations (e.g., "Melbourne trains", "Sydney trams"). The API access token is securely stored via Kubernetes Secrets and injected into the function at runtime.

- **Keyword filtering**

  Mastodon does not have a subreddit structure like Reddit, so we simulate topic relevance by keyword filtering:

  - Strong keyword: must appear at least once (e.g., myki, tram, vline)
  - Weak keyword: optional context words (e.g., delay, commute, overcrowded)
  - Forbidden keyword: exclude false positives (e.g., ranked, run, road, weekend)

Posts are only kept if (strong keyword + weak keyword) ≥ 2 and there are no forbidden keywords. All filtering logic is handled by the `match_topic()` function.

- **Text Cleaning**

  Each blog post content is stripped of HTML tags using `BeautifulSoup`, converted to lowercase, and noise characters and emoticons are removed. This improves the accuracy of keyword filtering and sentiment analysis.

- **Deduplication**

  To avoid indexing the same post repeatedly, each document is assigned a unique `_id` by taking the MD5 hash of the article URL. The function checks if the document already exists in the index to prevent duplication.

- **Store Elastic**

  Once a post passes all filters and gets a score, it is sent to the internal microservice endpoint `http://router.fission/addelastic`, which inserts the post into the `mastodon_v2` index.

4.6 ElasticSearch

The data is indexed in an in-cluster ElasticSearch deployment. In addition, a new data view is created specifically for transportation datasets and named `trans-reddit`.

- **Trans-reddit-sentiment**: Stores sentiment results for transport topics, posts, and comments.

- **Data Schema (Mappings)**:

```
doc = {"type": postType,
       "platform":"Reddit",
      "city": city.lower(),
      "sentiment":sentiment,
      "text": text,
      "upvote":post.score,
      "createdOn": datetime.fromtimestamp(post.created_utc).isoformat(),
      "url":post.url,
}
```

Fig

Figure 30:

4.7 RESTful API Design and Data Visualisation

This project uses RESTful API design to achieve data transmission between users and remote resources, so that Jupyter Notebook can easily access backend data. To simplify front-end access, the API does not have authentication, and all requests are completed through the standard GET method.

Jupyter Notebook is responsible for data analysis and visualisation. It does not directly connect to ElasticSearch but obtains data through the back-end service deployed on Kubernetes. The backend service handles all interactions with ElasticSearch, including receiving Notebook requests, building Query DSL queries, performing data retrieval and aggregation, extracting key fields and calculating statistical indicators (such as average sentiment score, user growth, etc.), and finally encapsulating the results as JSON and returning them.

5 Scenarios

To fully demonstrate the functions of this system, we designed and implemented two main analysis scenarios, focusing on sports social media analysis and urban public transportation public opinion analysis. Each scenario starts from the actual social issues related to Australia, combined with the cross-platform data collection, storage, analysis and visualisation capabilities provided by the system, and presents the research results through chart results.

5.1 Scenario 1: AFL social media analysis

5.1.1 Scenario description

This scenario focuses on Australia's most popular sports league—AFL. We use the data collected by the system from two mainstream social platforms, Reddit and Bluesky, to conduct a multi-dimensional analysis of AFL-related discussions, hot topics, fan emotions, the number of fans and their changing trends, and reveal the potential correlation between fan emotions and team performance. 18 teams were analysed and studied, focusing on the top 5 best performing teams and the bottom 5 worst performing teams, in order to find more significant comparisons and patterns.

This scenario includes the following sub-analysis tasks:

- **Overall analysis of cross-platform posts**: Compare the number, popularity and engagement of AFL posts on Reddit and Bluesky.
- **Cross-platform sentiment score comparison**: Compare the total sentiment scores of the top 5 and bottom 5 teams to show the differences in fans' sentiment attitudes between platforms.
- **Cross-platform topic attention comparison**: Identify the differences in the focus of users' discussions on AFL on Reddit and Bluesky.
- **Analysis of the relationship between sentiment trends and game results (by week)**: Compare the relationship between the weekly sentiment fluctuations of the top/bottom teams and the results of the game after 2024.
- **Analysis of the relationship between sentiment trends and winning rate (full year 2024)**: Analyse whether the sentiment trend and winning rate of the top/bottom teams in 2024 are related.
- **Home advantage analysis**: Explore whether there is a "home advantage" effect in AFL games.
- **Comparative analysis of the number of subscribers**: Count the number of fans on each platform and analyse the fan size of each team.
- **Dynamic analysis of sentiment and number of fans (after May 1, 2025, by day)**: Monitor the synchronization of daily sentiment changes and the growth of the number of fans.

5.1.2 Why choose this scenario?

AFL has wide attention and high commercial value. Fan activity and social media emotions are important indicators for evaluating team market performance. By analysing AFL data, the system can verify its cross-platform processing and time-series sentiment analysis capabilities, revealing the potential correlation of "game performance affects emotional changes and affects fan dynamics," and providing support for business decisions.

In addition, it is currently the AFL game season, which provides us with rich and fresh real-time data. Some project team members have also visited the scene to watch the game and have a strong interest and understanding of the AFL league, which enables us to be more insightful and targeted when designing analysis dimensions and interpreting analysis results.

5.1.3 Chart result analysis

- **Overall analysis of cross platform posts**

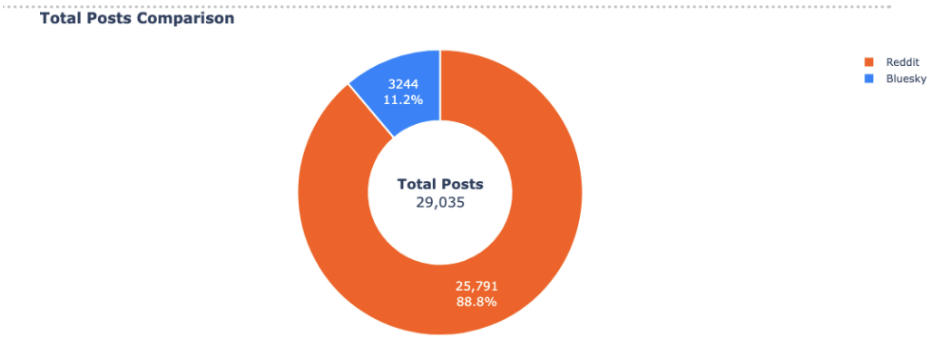


Figure 31:

The data shows that the volume of AFL-related discussions on Reddit is much higher than that on Bluesky, with a total of 25,791 posts collected, while Bluesky only has 3,244. This shows that Reddit is a more important gathering and discussion platform for AFL fans.

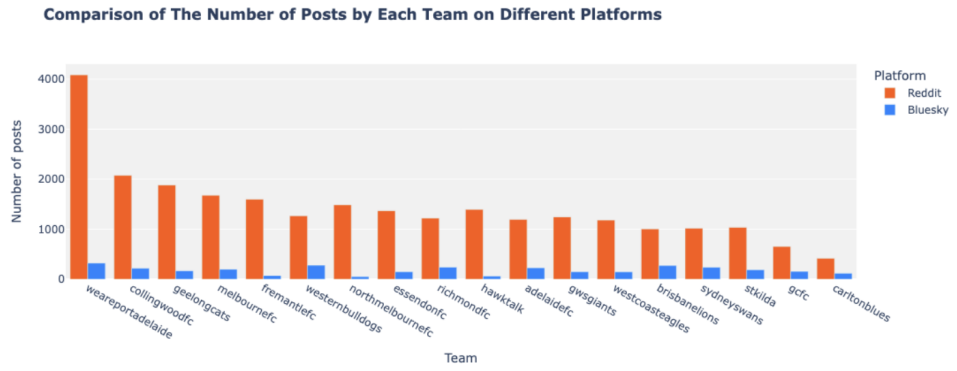


Figure 32:

On Reddit, Port Adelaide has the highest number of discussions, reaching 4,805 posts, far exceeding other teams. The total number of discussions on the Bluesky platform

is less, and the distribution is slightly different. Port Adelaide still ranks first, but its lead has narrowed. Western Bulldogs, Richmond, Brisbane Lions and Collingwood all have more than 200 posts, and Carlton is still at the bottom. Overall, Reddit is the main AFL discussion platform, with significantly higher discussion volume than Bluesky. The two platforms have roughly the same team discussion rankings, but the gap ratios vary, reflecting the differences in activity and preferences of fans of different teams on the platform.

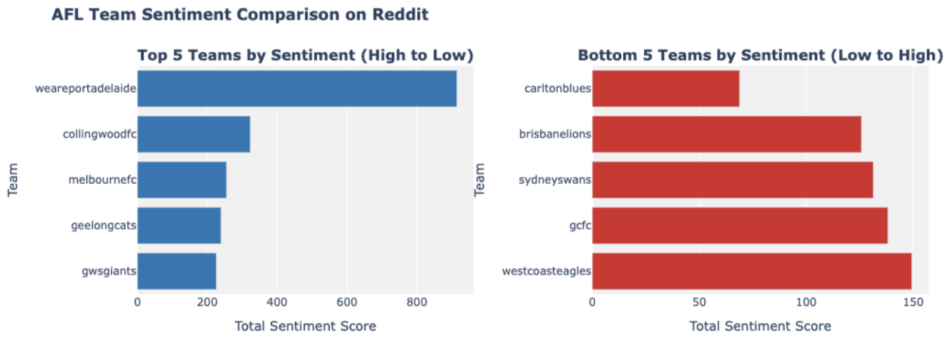


Figure 33:

Data analysis shows that the total sentiment value of the Reddit platform shows obvious echelon differentiation. Port Adelaide leads with 914 points, nearly 2.8 times the second place Collingwood (323 points), showing strong community emotional cohesion. At the tail, Carlton ranked last with 69 points, only 7.5% of the top, which may reflect significant problems in its fan relations. Traditional team Brisbane Lions (126 points) and Sydney (131 points) unexpectedly fell into the bottom five, while the old team West Coast (149 points) also performed below expectations.

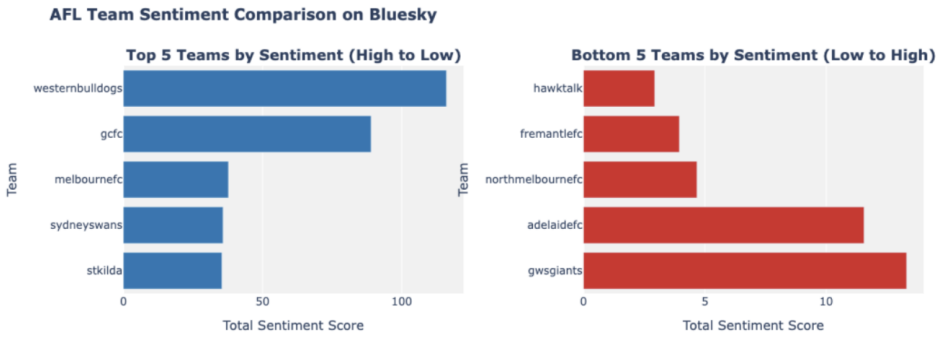


Figure 34:

In contrast to Reddit, the sentiment pattern of the Bluesky platform has changed. Western Bulldogs became the emotional champion. The tail is dominated by outliers, with traditional teams such as Hawthorn (3 points) and Fremantle (4 points) approaching zero sentiment.

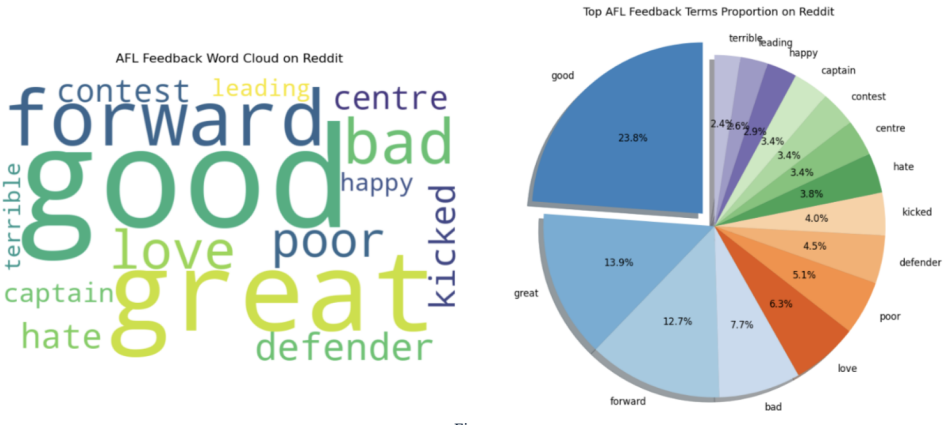


Figure 35:

The language characteristics of the Reddit platform show a significant emotional distribution, with positive sentiment words ("good" 23.8% + "great" 13.9% = 37.7%) dominating, but negative sentiment words are also concentrated ("bad" 7.7% + "poor" 5.1% + "terrible" 2.4% = 15.2%). Technical terms such as "forward" (12.7%), "defender" (4.5%) and "contest" (3.4%) also account for a high proportion (20.6%), indicating that Reddit users pay more attention to the technical aspects of the game when discussing AFL.

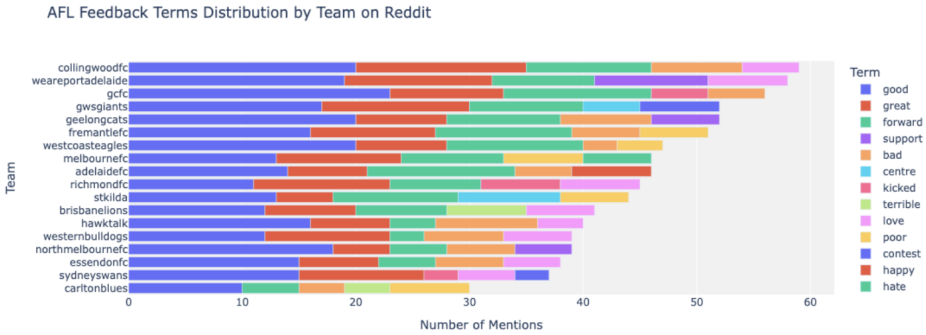


Figure 36:

In terms of team association, Port Adelaide has the highest proportion of technical discussions, with the mention of "forward" accounting for 19.3% of the team's total word frequency. Collingwood has the strongest emotional expression, with "love" and "hate" accounting for a total of 12.8%. It is worth noting that Carlton's negative word frequency reached 28.7%, far higher than the league average of 15.2%, suggesting that the team may face more negative comments on Reddit.

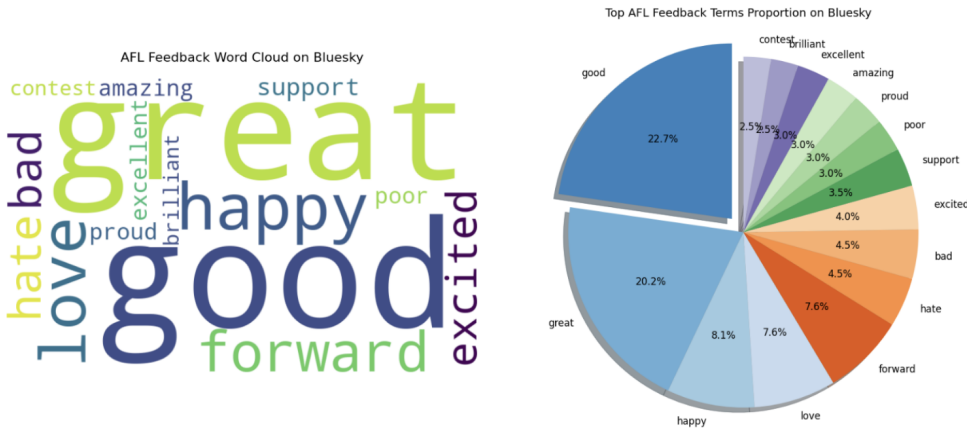


Figure 37:

Compared with Reddit, the Bluesky platform has a higher positive sentiment intensity, with "great" (20.2%), "excellent" (3%) and "brilliant" (2.5%) accounting for a total of 25.7%. Emotional expressions are also richer, with "excited" (4%) and "proud" (3%) accounting for a total of 7%. However, technical discussions are relatively rare, with "forward" (7.6%) and "contest" (2.5%) accounting for only 10.1

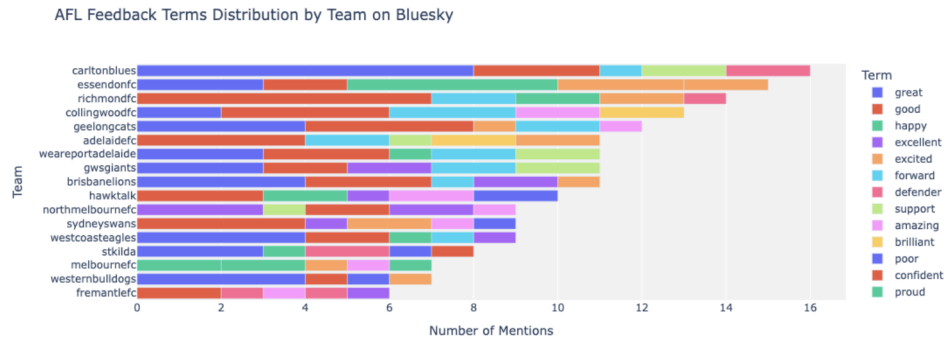


Figure 38:

In terms of team association, Western Bulldogs' positive words account for as high as 61.3%, showing that the team has a very high positive sentiment tendency on Bluesky. Fremantle has the highest technical word frequency, with "forward" and "defender" accounting for a total of 15.8%. Similar to Reddit, Carlton still maintains a negative tendency on Bluesky, with "poor" accounting for 8.9%.

In summary, the language characteristics of AFL related discussions on Reddit and Bluesky show obvious differences. Reddit users are more inclined to engage in technical discussions and express negative emotions more directly, while Bluesky users are more inclined to express positive emotions and identity, and technical discussions are relatively rare.

- **Analysis of the relationship between sentiment trends and game results (by week)**

This analysis explores the relationship between the total weekly sentiment of AFL teams

and the outcome of games on two platforms, focusing on the top two and bottom two teams in terms of sentiment in the second half of 2024 and early 2025.

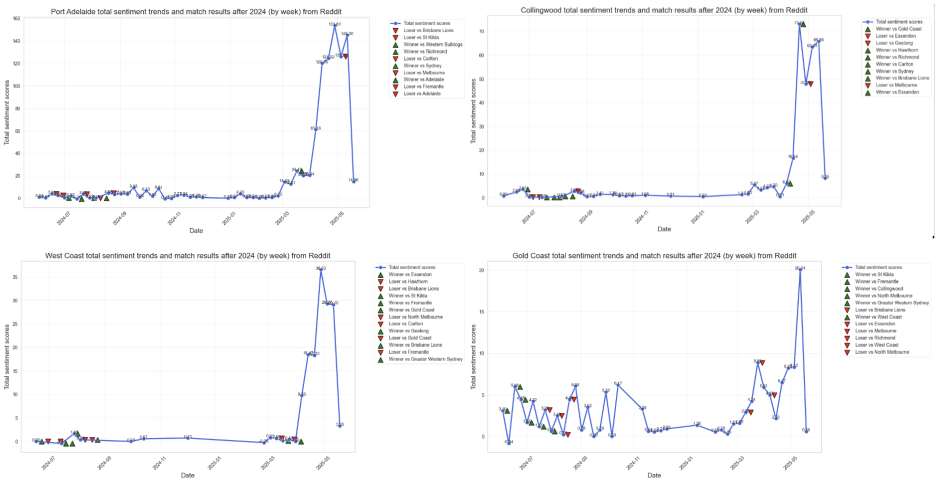
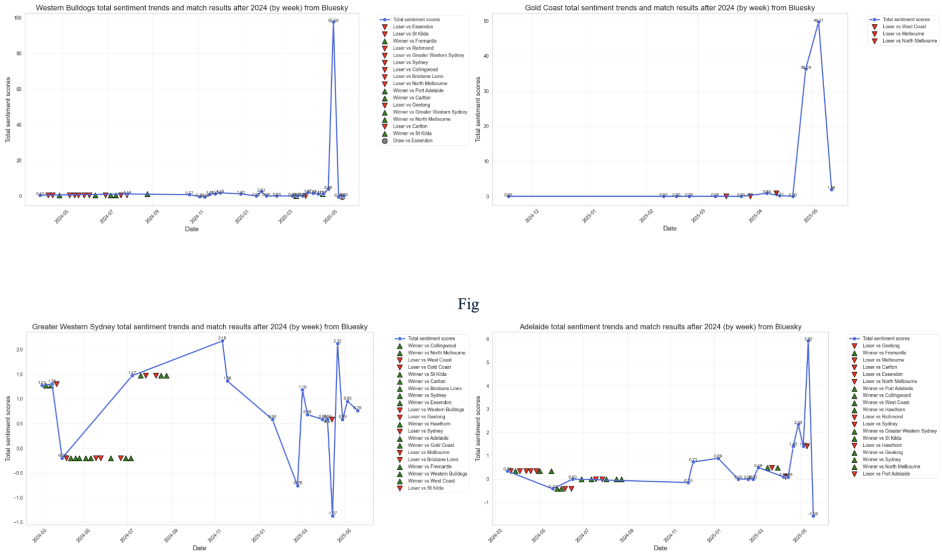


Figure 39:

In general, the weekly emotional fluctuations of AFL teams on Reddit are highly positively correlated with the results of the game, and victory significantly increases the positive emotions and discussion enthusiasm of fans. Emotional reactions vary from team to team: the emotional "outburst" is more obvious when the niche team wins, and the popular team can maintain a high emotional base even if it loses.



Fig

Figure 40:

In general, the emotional fluctuations on the Bluesky platform are weakly correlated with the results of the game, and the overall emotional fluctuation is smaller than that on Reddit. The victory or defeat of the game has a limited impact on user emotions, which may be due to the small user base, low activity or more restrained expression style.

- **Analysis of the relationship between sentiment trends and winning rate (full year 2024)**

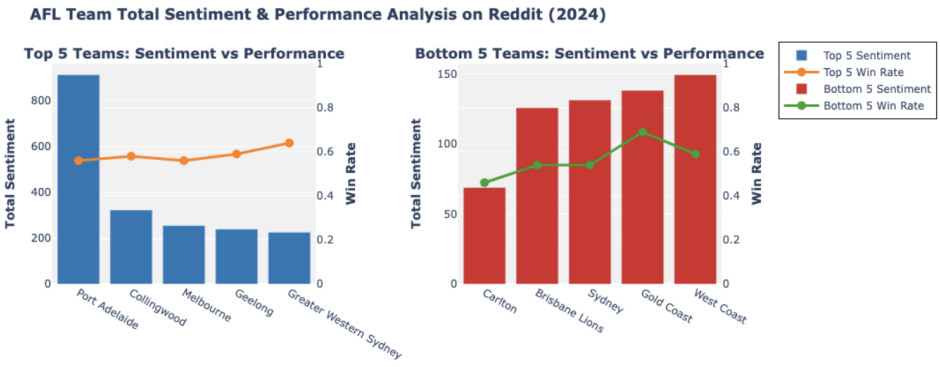


Figure 41:

Analysis of Reddit data shows that the average total sentiment of the top five teams is significantly higher than that of the bottom five teams, but the win rate is abnormal. For example, Gold Coast, which has a lower overall sentiment value, has an actual win rate (0.7) that even exceeds the average win rate of teams with higher sentiment scores. In addition, Port Adelaide's win rate does not show a significant advantage as its sentiment value does.

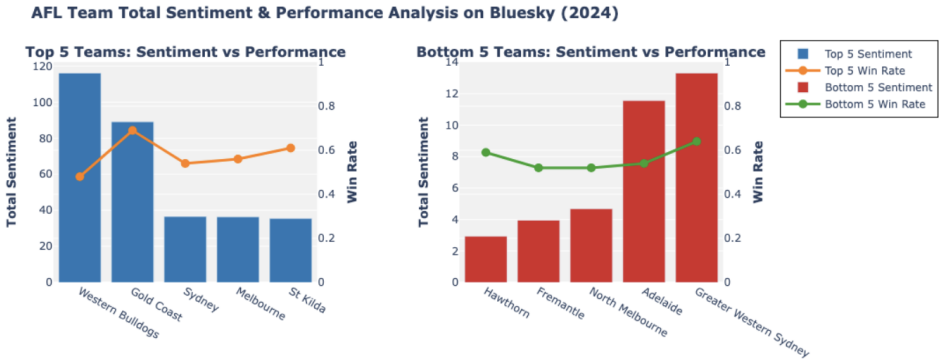


Figure 42:

Similar to Reddit, Bluesky also shows that there is no obvious correlation between sentiment and win rate. Western Bulldogs have the highest sentiment value, but its win rate is significantly lower than other teams, while Hawthorn has a very low sentiment value (3 points), but its win rate is 0.6, which is a huge contrast. Melbourne is the only team that is in the top 1/3 of both sentiment and win rate on both platforms, which may reflect the team's stable support and performance on and off the field.

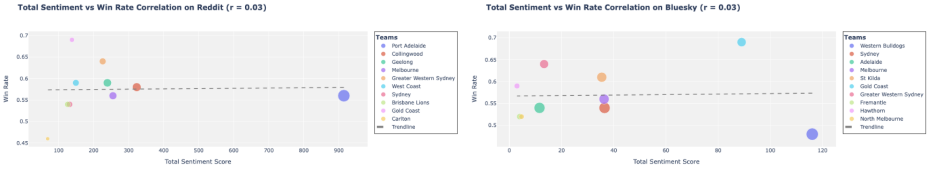


Figure 43:

The scatter plots of the two platforms show that the total emotional score has almost no correlation with the team's winning rate ($r = 0.03$). Emotional heat cannot predict team performance, and a high emotional score does not mean a high winning rate.

- **Home advantage analysis**

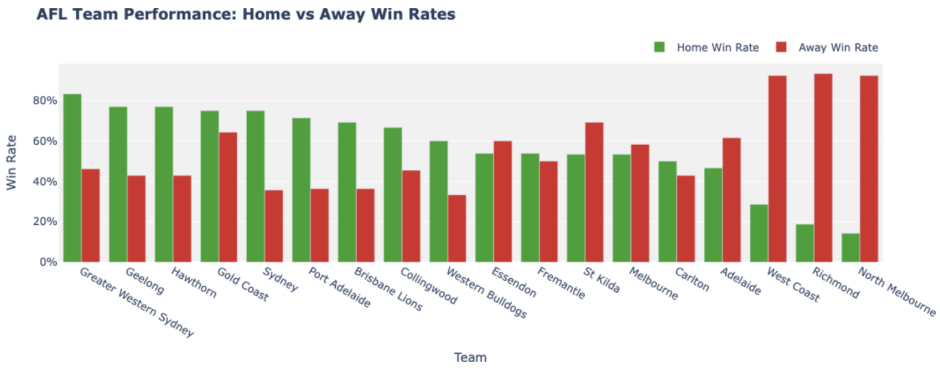


Figure 44:

Some teams have obvious home advantages, such as Greater Western Sydney, Geelong, and Hawthorn, with a home winning rate of 75%, while away games drop to about 40%. Sydney, Port Adelaide, and Brisbane Lions are also better at playing at home, reflecting the importance of the home environment and fan support. West Coast, Richmond and North Melbourne have an away win rate of nearly 90%, but their home performance is poor, indicating that they may have stronger tactical execution or adaptability on the road.

- **Comparative analysis of the number of subscribers**

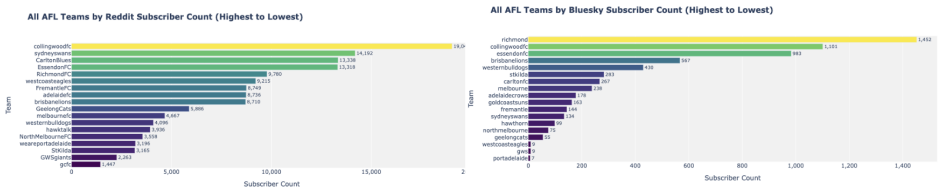


Figure 45:

In general, the huge difference in subscriber numbers between Reddit and Bluesky highlights Reddit's status as the leading online hub for AFL fan engagement, and it remains a key platform for fan interaction and community building. While some teams rank relatively consistently in popularity on both platforms (e.g., Collingwood and Essendon both rank highly), there are significant variations. For example, Richmond leads on Bluesky but is in the middle of the pack on Reddit.

- **Dynamic analysis of sentiment and number of fans (after May 1, 2025, by day)**

  This analysis monitors the synchronization of daily total sentiment changes and fan growth of the top two and bottom two teams in total sentiment on Reddit.

Fig

Figure 46:

Overall, there is a certain positive correlation between the total sentiment changes of AFL teams on the Reddit platform and the growth of the number of fans, but it is not completely synchronized and there is a certain time lag. Positive community sentiment generally indicates an increase in the number of fans, while negative sentiment may lead to fan loss or stagnation of growth. However, the strength and synchronization of this relationship vary from team to team, and fan growth is affected by multiple factors.

5.2 Scenario 2: Social Media Analysis of Public Transportation in Australian Cities

5.2.1 Scenario Description

This scenario conducts public opinion analysis on the public transportation systems of eight major Australian cities, including Sydney, Melbourne, and Brisbane. We collect posts on topics about public transportation (such as subways, buses, etc.) on Mastodon and Reddit, and compare user feedback from different cities for sentiment and identify hot topics.

This scenario includes the following sub-analysis tasks:

- **Cross platform overall post analysis:** Analyse the number of posts and active user distribution in each city.
- **Cross platform sentiment comparison analysis:** Compare the traffic sentiment scores of different cities on the two platforms to identify high/low satisfaction cities.
- **Cross platform topic attention comparison:** Identify the key differences in user discussions on urban transportation on Mastodon and Reddit.
- **Urban public transportation problem identification and trends:** Identify the most frequently mentioned public transportation problems and trends on social media.

5.2.2 Why choose this scenario?

Public transportation, as a key focus of urban governance, is closely related to residents' lives. Social media has become an important channel for public feedback. By analysing traffic discussions in different cities and platforms, the system can demonstrate its capabilities in multi-source data integration, cross platform comparison, and urban sentiment mining, helping the government optimize transportation services.

In addition, team members frequently encounter typical problems such as peak delays and card swiping failures when using Melbourne's Myki system and trams/trains daily. These actual experiences are consistent with the negative reviews of Melbourne public transportation, such as high fares and many delays, which prompted us to objectively verify the hypothesis of "whether Melbourne public transportation has the lowest rating in Australia" through social media big data.

5.2.3 Chart result analysis

- **Cross platform overall post analysis**

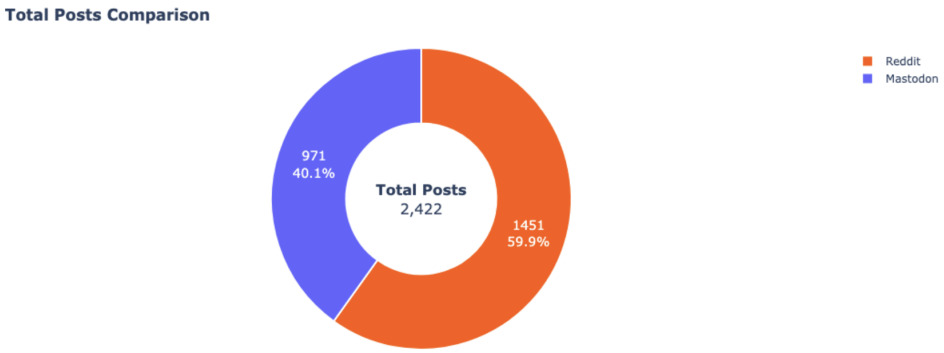


Figure 47:

Our ElasticSearch database contains 971 public transportation discussion data from Mastodon and 1451 from Reddit, which shows that Reddit is more active in discussing public transportation topics.

- **Cross platform overall post analysis**

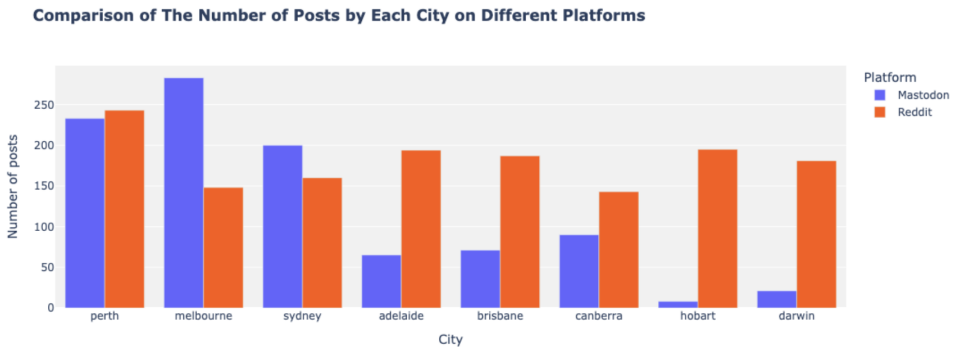
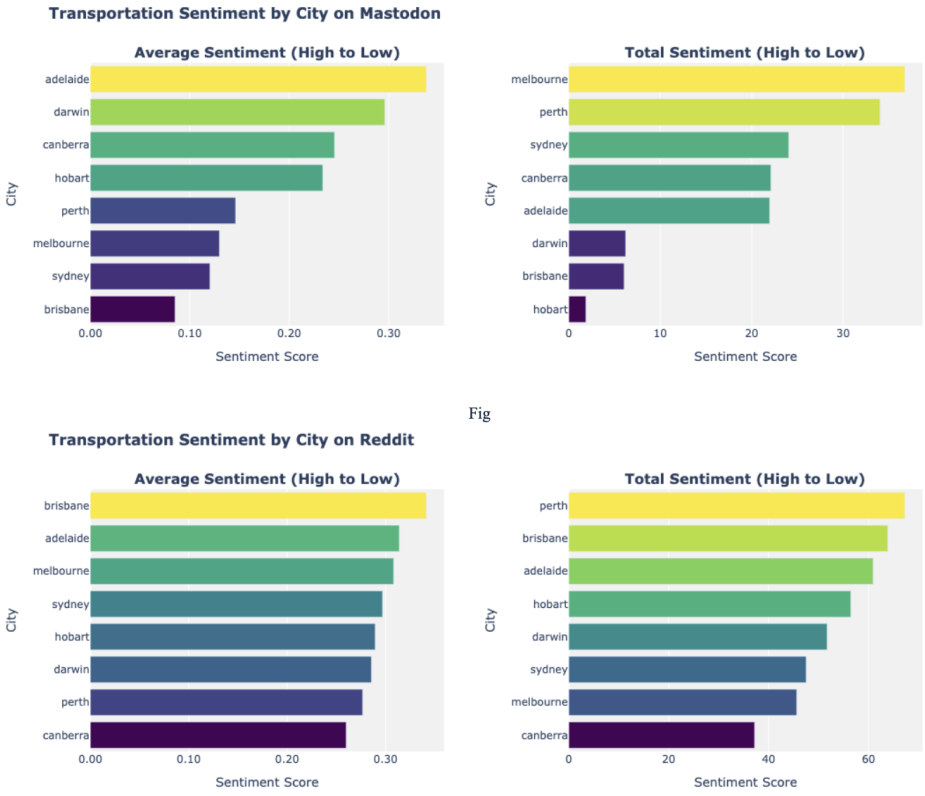


Figure 48:

In terms of discussion popularity in different cities, Mastodon users mainly focus on traffic conditions in Melbourne, Perth and Sydney. In contrast, Reddit users show a higher level of interest in traffic conditions in Perth, Hobart and Adelaide. An interesting phenomenon is that Hobart's traffic has hardly sparked discussion on Mastodon, but has become a hot topic on Reddit, which reveals the differences in the focus of user groups on different platforms.

- **Cross platform sentiment comparison analysis**



Fig

Figure 49:

Overall, users' sentiment evaluations of public transportation in the same city vary between different platforms. Melbourne has the largest contrast in evaluations on Mastodon (0.13) and Reddit (0.31), which may reflect the differences in focus of different user groups. Adelaide remains in the top two on both platforms, indicating that its public transportation services are recognized across groups. The reliability of the analysis results may be reduced in small cities such as Hobart and Darwin due to insufficient posts.

- **Cross platform topic attention comparison**

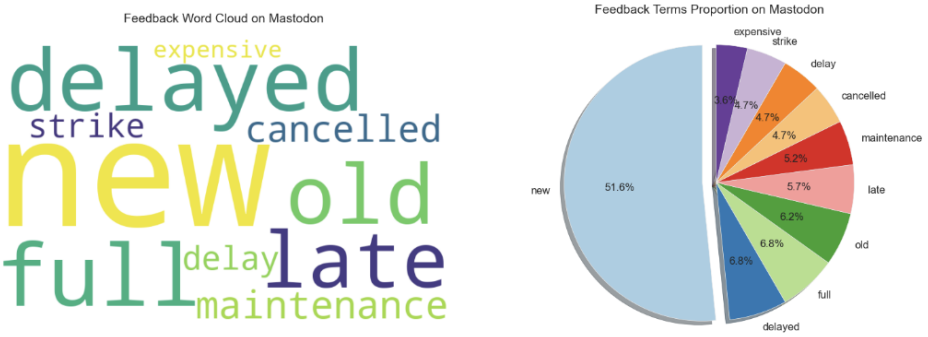


Figure 50:

On Mastodon, the keyword "new" ranked first in public transportation discussions with a significant advantage (51.6%), indicating that users are highly concerned about new facilities, routes and policies, which may be related to the renewal of transportation systems in some Australian cities. It was followed by a series of words reflecting negative experiences, such as "delayed", "full", "old", "late", "maintenance", "cancelled", "delay" and "strike" (about 4–7%), indicating that punctuality, congestion and service interruptions are the focus of users. Although "expensive" also made it into the top ten, it received less attention.

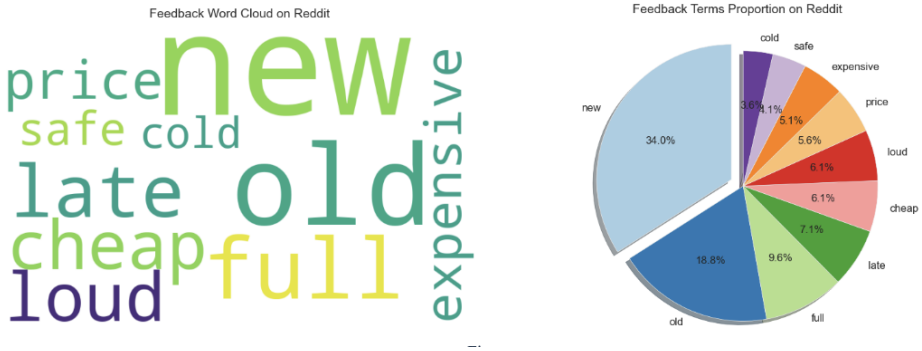


Figure 51:

On Reddit, "new" also ranked first (33%), but the proportion was lower than that of Mastodon. The second was "old" (18.8%), indicating that users are more concerned about the aging of the system and the need for renewal. "Full" and "late" were also prominent, with a trend similar to Mastodon. Reddit's specific keywords include "loud", "safe", and ticket related "cheap", "price", and "expensive", reflecting a more diverse range of user views.

Overall, users of both platforms are concerned about "new developments", but Reddit is more concerned about aging, safety, and ticket price diversity, while Mastodon is more focused on new projects and service interruptions.

- **Urban public transportation problem identification and trends**

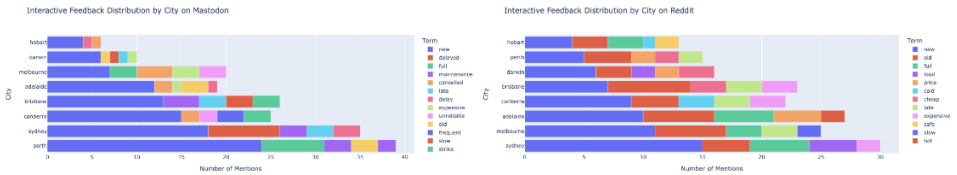


Figure 52:

Analysis of urban transportation shows that Mastodon and Reddit users have both similarities and differences in their concerns. "New" is commonly mentioned in various cities, reflecting the concern about infrastructure renewal. Melbourne and Sydney have consistent performance across platforms on operational issues such as "delay", "late" and "cancelled". Reddit is more concerned about fares, especially in Brisbane and Canberra; Mastodon places more emphasis on safety and service interruptions, such as "dangerous" in Darwin and "broken" in Adelaide. In addition, "old" frequently appears on Reddit, indicating concerns about aging facilities. In terms of city characteristics, Melbourne often sees "full" and "slow" on both platforms, highlighting congestion; Sydney's "loud" on Reddit points to noise problems.

6 Error Handling

In this project, we encountered several significant challenges and issues related to **Kubernetes**, **Fission**, and **ElasticSearch**. This section details these problems and outlines the solutions we implemented to overcome them.

Error Scenario 1: Fail to create a package

a. **Problem Description:**
When creating a Fission package, users often see error:
"Error uploading deployment package: Internal error - error archiving zip file: file already exists" or "./build.sh: no such file or directory"

b. **Causes:**

- **Line Endings:**
Build scripts must use Unix line endings ("POSIX shell script, ASCII text executable"), not Windows line endings ("POSIX shell script, ASCII text executable, with CRLF line terminators").
- **File Permissions:**
The builder needs read and write access to the `build.sh` file. Wrong permissions can stop zipping or running scripts.

c. **Fix:**

- **Check Permissions:**
Do not use root to create `build.sh` and requirements. Ensure all files belong to a normal user (use `ls -l` to check). Also, grant shell scripts (build.sh) the "executable" permission (use `chmod +x build.sh`).

- **Check Line Endings:**
  Use the command `file build.sh` to inspect line endings. If the output shows "POSIX shell script, ASCII text executable, with CRLF line terminators", use `dos2unix build.sh` to fix it.

Error Scenario 2: Fission Runtime Timeout from `atproto` Import

a. Problem Description:
In Fission, importing the `atproto` package causes the function to fail at runtime — even if the import is unused. The package and function can be created and built successfully, and no errors occur during the build phase. However, when running `fission fn test`, the function always times out, and logs show the process hanging during the specialization phase (e.g., `specialize called`), eventually resulting in a deadline exceeded error.

Figure 53:

b. Causes:
Fission's default Python environments are based on minimal Alpine Linux images, which often lack essential native system libraries such as `libffi`, `openssl`, and `libc`. These libraries are required by packages like `cffi`, `cryptography`, or `atproto`, either at build time or during runtime. As a result, functions that depend on these libraries may fail to build properly or hang during execution — especially if the package performs system-level initialization (e.g., socket setup, DNS resolution, or threading) when imported.

c. Fix:
The issue was fixed by removing the `atproto` package and using the `httpx` library to make direct HTTP requests to Bluesky's API. This avoids the import-time blocking and native dependency issues caused by `atproto`, which are incompatible with Fission's default Python environment. By calling the API endpoints directly (e.g., for login and profile retrieval), the function now runs reliably without requiring custom Docker images or native library support.

Error Scenario 3: Fission Function Routing Timeout

a. Problem Description:
When accessing a deployed Fission function via `curl http://localhost:8888/mastodon`, the following error occurs:
error sending request to function

b. Causes:
The harvesting function timed out and did not respond. After 5 minutes, the port automatically stopped listening.

c. Fix:
Our goal is to reduce the function running time and remove some of the functions. We wrote separate functions to implement some common tasks such as getting posts from the API, uploading to elastic search, and text cleaning, and then called them from within Fission. This effectively reduces the function running time and improves resource utilization.

Error Scenario 4: ElasticSearch Connection and Query Failures

a. Problem Description:
When querying AFL team sentiment data from ElasticSearch, if the query is not successfully executed due to authentication failure, connection configuration error, or query syntax problems, connection exceptions (such as `ConnectionError`, `AuthenticationException`) or runtime errors may be triggered.

b. Causes:

- ElasticSearch requires username and password authentication. If the key file path is incorrect or the file is empty, authentication will fail.
- If the ElasticSearch service address is incorrect or the cluster is unavailable, the connection will fail.
- If the field name in the query syntax does not match the actual index structure, the query may fail.

c. Fix:

- Use `verify_certs=False` and `ssl_show_warn=False`: Skip certificate verification in cluster internal communication to avoid errors due to incomplete SSL configuration.
- Print `key_operation` logs through `current_app.logger.info(...)`. This log is output before the query is executed. If the program fails later, it can help locate whether the problem is caused by the query execution process, which is helpful for debugging and error tracking.

Error Scenario 5: Malformed API Response Handling

a. Problem Description:
When calling the Fission API `http://localhost:8080/afl/sentiment/reddit`, if the returned JSON data lacks the expected fields (such as `top_5_teams_reddit`), subsequent data processing (such as constructing a DataFrame or sorting) will cause program errors, such as throwing `KeyError` or `TypeError`.

b. Causes:
The API may return incomplete or erroneous data without the expected fields due to incorrect request parameters, abnormal backend logic, empty data source, or unresponsive service.

c. Fix:
Before processing the response, check whether `top_5_teams_reddit` exists in the returned JSON to make an error judgment. If the field is missing, print the error message, and give priority to outputting the error field content returned by the backend. If there is no such field, return the default error prompt `"Unknown data error"`. This can avoid program crashes and provide effective information for debugging.

Error Scenario 6: Fission Function Internal Timeout

a. Problem Description:
While invoking the Fission function `mastodon-harvester`, the following error occurred:
`Error executing HTTP request: Get "http://127.0.0.1:60892/fission-function/masto`
`function request timeout (60000000000ns) Exceeded`

b. Causes:
This may be because the function has already been executed once while the Pod is still cold started, which causes unresponsiveness when the Pod is ready to call the function again. It may also be because the harvesting function depends on some external resources such as `nltk` every time it runs.

c. Fix:
Avoid loading large resources such as `nltk` at runtime, but use `nltk.download` to pre-bundle. Secondly, set a sleep time inside the main function to ensure that the function is run after the Pod cold start is completed.

Error Scenario 7: 500 Internal Server Error

a. Problem Description:
When calling the Mastodon harvesting function through Fission's HTTP gateway, it returns:

Figure 54:

b. Causes:
This is because an error occurred during the runtime of my function. The general error may be that the return format is wrong, Flask needs a dictionary or string return. Or a KeyError occurs, and the attempt to access the Mastodon API is rejected. It may also be that the TOKEN for accessing the API fails, the secret is not mounted correctly, or the secret cannot be read.

c. **Fix:**
Use unit tests to verify that each function component runs as expected, including correct token retrieval. After verifying all parts individually, assemble and test them together. Add `try-except` blocks to catch abnormal values and errors, making it easier to identify root causes when reviewing logs.

Error Scenario 8: Docker Image Incompatible with Fission

a. **Problem Description:**
A Fission function was implemented to retrieve the number of Facebook followers for various AFL teams using Selenium and Chromium. The function utilized `selenium.webdriver` with Chrome options and was packaged in a Docker image based on `fission/python-env:3.9`. Although the function and environment were successfully registered within the Fission framework, all test invocations resulted in timeouts. Diagnostic output showed no active pods being created for the function, indicating a failure at the cold start stage.

b. **Causes:**
The custom Docker image was built on an ARM64-based machine (MacBook M3), resulting in an ARM64-architecture container. However, both chromium and chromedriver installed via apt are compiled for x86_64 (Intel) architecture. This mismatch caused incompatibility between the browser and the driver, preventing proper execution in the ARM64 environment and also making the image unusable on standard x86_64 Kubernetes nodes.

c. **Fix:**
Instead of continuing with the Chromium-based scraping setup, the approach was changed to extract AFL follower data from Bluesky. This method avoids the need for a browser automation environment and eliminates the dependency on platform-specific binaries, ensuring compatibility and stability within the Fission deployment pipeline.

7 Fault Tolerance

We were able to create a fault tolerant system by utilising **Kubernetes** and independent **Fission** functions to reduce the risk of systemic failure. Kubernetes provides automatic pod recovery, ensuring that failed services are restarted without manual intervention. In our NeCTAR cluster we also deployed multiple virtual machines which allow to avoid relying on a single node, further enhancing its fault tolerance.

Each social media harvester (Reddit, Bluesky & Mastodon) is implemented as an independent serverless function using Fission. This event-driven design ensures that if one harvester fails, others will continue to operate without interruption, preventing a complete system outage. Redis was employed as a message queue to decouple ingestion from processing, buffering incoming tasks during traffic spikes and enabling asynchronous execution of Fission functions to prevent timeouts.

Redis and Elasticsearch could also represent potential single points of failure due to Redis's in-memory nature being vulnerable to data loss and a failure in the Redis server would directly impact functions (harvester in this case) that rely on it for message

queuing, potentially causing them to fail or lose queued tasks. Similarly, Elasticsearch's indexing operations can become a bottleneck under high ingestion rates, especially when handling continuous streams of social media data. This can increase latency and strain cluster resources if not configured properly. We were able to mitigate these risks by using Kubernetes health checks and restart polices to monitor and recover if it does fails. This combination of serverless architecture and orchestration impro ves the system's resilience and fault tolerance.

KEDA contributes to fault tolerance by automatically scaling Fission functions based on Redis queue depth. Even if functions are scaled down to zero during idle periods, KEDA ensures they are restarted when demand increases. This allows us to enhance system availability and robustness, making it capable of recovering from partial failures and maintaining functionality under varied load conditions.

8 Links

8.1 Link to YouTube

Here is the YouTube video list for all the videos: **COMP90024 - Playlist**

8.2 Link to Gitlab

Here is the link to our Gitlab: **COMP90024-TEAM54**

9 Roles of Team Members

NAME	STUDENT ID	Roles
Xueying Yuan	1531588	Frontend
Shuangquan Zheng	1331085	AFL Reddit - Backend
JunJie	1103084	Transport Mastodon - Backend
Hao Xu	1468277	Transport Reddit - Backend
Chunmiao Zheng	1642700	AFL Bluesky - Backend

10 Conclusion

Based on the NeCTAR Research Cloud, this project successfully designed and implemented a scalable cloud system analytics platform for processing and visualising social media data from Reddit, Bluesky, and Mastodon. Through Kubernetes management, Fission serverless functions, and ElasticSearch real-time data storage, we built a powerful system that supports dynamic scaling, fault tolerance, and cross-platform sentiment analysis.

Our main achievements include:

- **Efficient data processing:** The modular architecture supports flexible integration of new data sources such as AFL teams and urban traffic topics, while reusing shared functions such as text cleaning and sentiment analysis.

- **Dynamic scaling:** Using KEDA and Redis, we implemented event-driven resource allocation to effectively cope with NeCTAR's quota restrictions and traffic spikes.
- **Cross platform discussion:** Rich comparative analysis reveals differences in user behaviour across platforms, such as Reddit users being more active and expressing more positive emotions.

During the project implementation, we also successfully solved many challenges through iterative debugging and architecture optimization, including Kubernetes Pod eviction, Fission cold start delays, and social media platform API failures. These improvements ensured the stability and high performance of the entire system.