COMP90024 Team 75 (2025), **xx**:xx 1–25

Project Report

Social Conversations on Housing Affordability Analysis

Linying Wei (1638206), Wangyang Wu (1641248), Ziyu Wang (1560831) and Roger Zhang (1079986)

Keywords: housing affordability, social-media mining, Reddit, Mastodon, Kubernetes

Abstract

In this report, we implement a cloud-native, serverless pipeline for monitoring public discourse on housing affordability through real-time analysis of Reddit and Mastodon posts and comments. Our system uses Kubernetes-orchestrated Fission functions to harvest social-media data at regular intervals, processes text with HTML stripping and noise filtering, and applies VADER sentiment scoring before indexing documents into Elasticsearch. A RESTful API and interactive Jupyter interfaces are provided to explore time-series trends, sentiment distributions, and geographic breakdowns. Multi-dimensional analyses reveal that social-media sentiment reliably mirrors real-world variations in housing affordability pressures. Our project offers a fine-grained, near-real-time complement to traditional surveys and economic indicators. Future work will focus on CI/CD pipeline and noise filtering refine to enhance topic coherence.

Video Demonstration Link
Gitlab Link

Contents

1. Introduction

1.1. Project Background and Motivation

In recent years, housing affordability has become one of the most pressing concerns across Australia. While governments regularly publish formal reports, much of the public's real-time sentiment and lived experiences are captured in online conversations, especially on social platforms such as Reddit and Mastodon. However, extracting meaningful insights from this noisy and unstructured data is challenging due to the informal language, lack of metadata, and high volume of content. Moreover, traditional survey measures often suffer from time lags and limited geographic resolution while social media offers high-frequency posts that can reveal emerging affordability pressures at the state level.

Hu et al. (2019)[9] integrated machine-learning algorithms with hedonic price modeling to produce high-resolution rental price monitoring maps from social-media sources. We recognize that a similar framework can be applied to sentiment analysis. By combining NLP-derived sentiment indicators with a cloud-native, serverless data pipeline, we can generate real-time, state-level housing affordability sentiment metrics across Australia, thereby overcoming the time-lag and spatial-resolution limitations of traditional surveys and providing domain experts with timely, fine-grained sentiment data.

1.2. Application System Objectives

This cloud-based application is designed to continuously harvest high-volume discussions about housing affordability from Mastodon and Reddit using Kubernetes-scheduled serverless functions, process the incoming content through a modular pipeline that performs text cleaning, sentiment scoring, and index the results into Elasticsearch to enable rapid and real-time queries. Through a RESTful API, clients can retrieve sentiment trends, top keywords and bigrams, and discussion volume metrics, while embedded Kibana dashboards and Jupyter Notebook templates support interactive visualization and ad-hoc analysis. The entire system is built with infrastructure-as-code, retry logic, and independent monitoring for each service, ensuring resilience, extensibility to new data sources or analytic modules, and low operational overhead.

2. System Architecture and Design

2.1. System Architecture Overview

The system integrates virtualization, containerization, orchestration, and serverless computing technologies (see Figure 1). We use the NeCTAR Research Cloud to host our infrastructure, deploying four virtual machines to support system components. Each service is encapsulated using Docker containers to ensure portability and consistency across environments. Kubernetes serves as the orchestration layer, managing container scheduling, scaling, and communication.

2.2. Component Overview

2.2.1. Reddit / Mastodon Harvesters

The data harvesting layer implements a multi-platform architecture for collecting housing-related discussions from Reddit and Mastodon. It separates development from production deployment, enabling scalable and testable ingestion workflows.

For development and testing purposes, lightweight scripts interact directly with platform APIs to validate functionality and collect sample data. In production, Mastodon harvesting supports both real-time and historical modes: real-time data is collected using Fission-based functions deployed on Kubernetes, while historical harvesting over a five-year range is handled via scheduled batch jobs.

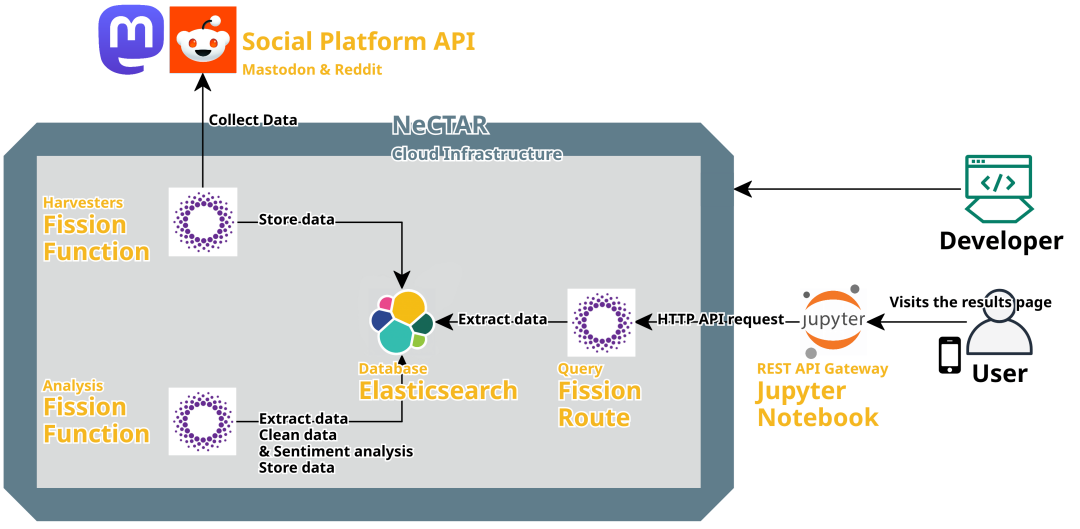


Figure 1: System Architecture

Reddit harvesting adopts a producer-consumer model using Kubernetes CronJobs to generate harvesting tasks and Fission functions to execute them. Task distribution is managed via Redis Streams, supporting parallel execution and retry logic. All harvesters incorporate metadata tagging and error-handling strategies to ensure consistency and reliability across platforms and timeframes.

2.2.2. Data Ingestion through Fission Functions

Our system implements data collection from Reddit and Mastodon platforms using Fission functions. The deployment of these functions is managed through a structured process that includes packaging the backend code and its dependencies into a deployment package. The functions are scheduled to run periodically using Fission's timer mechanism, ensuring regular data collection and processing.

The deployment process involves preparing the deployment package, deploying it to the Fission environment, and scheduling the functions to run at regular intervals. This setup ensures that data collection is automated and reliable, with minimal manual intervention required.

2.2.3. Elasticsearch Storage and Query Engine

The collected data is stored in Elasticsearch with a well-defined schema that includes fields for post identification, content, metadata, and analysis results. The system maintains separate indices for posts and comments, with standardized fields including post ID, platform, author, content, created timestamp, state, topic, and sentiment score.

The Elasticsearch configuration is defined through index mappings that specify field types and analysis settings. This structured storage enables efficient querying and analysis of the collected data, supporting various analytical needs such as sentiment analysis and topic modeling.

2.2.4. API Layer and Jupyter Notebook Frontend

Our API layer is implemented as lightweight Fission HTTP routes that accept parameters perform Elasticsearch queries or scans, and return JSON, eliminating the need for a separate web framework. The frontend consists of a single Jupyter Notebook that fetches these JSON endpoints through Python's

`requests`, loads the results into pandas DataFrames, and presents a suite of interactive visualizations, such as time-series plots, bar charts, word clouds, n-gram and topic tables. We also embed Kibana dashboards for real-time monitoring.

2.3. Module Interactions and Data Flow

2.3.1. Data Pipeline

Our data pipeline consists of a modular, serverless workflow that automates the ingestion, processing, and storage of social media data. As illustrated by the arrows in Figure 1, the process starts with Python-based harvesters deployed as Fission functions that collect posts and comments from social platforms, Reddit and Mastodon, via their public APIs.

These raw JSON objects are then cleaned, and passed to subsequent Fission functions responsible for sentiment analysis using VADER. Once attached with sentiment score, the data is forwarded to an Elasticsearch instance, where it is indexed under separate collections for posts and comments. This makes the data easily searchable and filterable for downstream applications.

Finally, users and developers interact with the data using HTTP API request on Jupyter Notebook, which enables exploratory analysis, query execution, and visualizations. This modular and event-driven pipeline ensures that each step in the process is scalable, loosely coupled, and easily maintainable.

2.3.2. Harvester-Storage Integration

The harvester modules integrate directly with Elasticsearch for efficient data indexing. Historical harvesters and scheduled batch jobs ingest data in bulk, leveraging optimized batch sizes to improve performance and reduce overhead. Real-time harvesters, implemented as Fission functions, maintain incremental state through dedicated control indices to support continuous ingestion without duplication.

To ensure ingestion reliability, all harvesters apply create-only indexing operations and incorporate retry mechanisms for transient failures. Indexed data is organized by platform, type (post/comment), and temporal range, enabling fast querying and clear separation between real-time and historical content.

2.3.3. Task Orchestration Flow

The system employs distinct orchestration strategies for Reddit and Mastodon to manage data harvesting efficiently.

For Reddit, a producer-consumer architecture is used. A scheduled component periodically generates harvesting tasks based on dynamically computed time windows and queues them in Redis Streams. These tasks include metadata such as subreddit names and keywords, and are consumed asynchronously by Fission functions, allowing for parallel processing and fault-tolerant execution.

Mastodon harvesting combines two orchestration modes. Real-time data is collected by scheduled Fission functions that maintain incremental state using Elasticsearch control indices, enabling collection of only new posts since the last execution. Historical data collection is handled by independent Kubernetes batch jobs, each configured to process specific hashtags across a predefined timeframe.

Across both platforms, orchestration workflows rely on Kubernetes ConfigMaps for secure credential management and Elasticsearch for persistent state tracking. This modular design ensures that task generation, scheduling, and processing remain decoupled and scalable, while maintaining reliable ingestion pipelines for both real-time and historical data.

2.4. Design Decisions and Rationale

2.4.1. Technology Stack Selection

The technology selection process was guided by the architectural requirements of distributed social media data harvesting, with an emphasis on scalability, fault tolerance, and deployment efficiency in

cloud environments. Each component was chosen after evaluating alternatives against project constraints and operational needs.

Kubernetes was adopted for container orchestration due to its robust support for heterogeneous workloads [4]. The need to manage long-running services (e.g., Elasticsearch, Redis) alongside batch jobs (e.g., historical Mastodon collection) made its native support for CronJobs and Jobs essential. Alternatives like Docker Swarm and Apache Mesos lacked integrated batch processing capabilities or required external tooling, increasing system complexity [1]. Kubernetes also offered mature tools for service discovery, configuration management, and resource control.

For serverless execution, Fission was selected over AWS Lambda, OpenFaaS, and Knative due to its native integration with Kubernetes and optimized Python runtime. This choice avoided vendor lock-in and ensured portability across different cloud providers. Fission's container-based model provided better control over cold starts and resource allocation, while preserving the stateless execution model required for fault-tolerant data ingestion.

Redis Streams was selected over basic Redis Lists due to its reliability and support for consumer groups. Its design addresses several architectural shortcomings through mechanisms such as explicit acknowledgment (`XACK`), pending entries list (PEL) for failure detection and recovery, horizontal task distribution across consumer group members, and rich observability for debugging and queue health monitoring[15]. These features significantly improved system resilience in a Kubernetes environment where pod restarts and preemptions are common.

Elasticsearch was chosen as the primary data store for its powerful full-text search and aggregation capabilities. Compared to general-purpose databases like MongoDB or relational systems, Elasticsearch offered real-time indexing, scalability, and a document-based schema that aligned with the analytical nature of social media content[11].

FastAPI was selected to build RESTful services, offering asynchronous request handling, automatic documentation generation, and strong typing support. Compared to Flask and Django, it reduced boilerplate code and improved development speed while better utilizing system resources for I/O-bound tasks[2].

2.4.2. Scalability vs. Fault Tolerance Tradeoffs

The system architecture balances scalability and fault tolerance by tailoring each subsystem to its operational role within the data pipeline.

Task distribution for Reddit harvesting emphasizes scalability through Redis Streams. A single-master Redis setup was selected to enable high-throughput task dissemination across Fission consumers, accepting the risk of central bottlenecks. This decision is mitigated by a dual-trigger mechanism: a Kubernetes CronJob periodically produces new harvesting tasks, while Fission functions are also scheduled independently via TimerTriggers to consume tasks, ensuring a consistent flow even if one scheduler experiences delays[6].

State management is decentralized using Elasticsearch control indices, supporting component-level scaling without a centralized store. This introduces eventual consistency and possible coordination conflicts, but such tradeoffs are acceptable given the system's analytical focus, where approximate completeness is sufficient for trend detection.

The processing layer adopts a stateless model via Fission, favoring horizontal scalability over execution continuity. Individual message loss is tolerated in favor of elastic scaling, enabling the system to respond effectively to variable data volumes.

Mastodon harvesting illustrates mode-specific priorities. Historical batch jobs emphasize reliability through retry mechanisms and persistent checkpoints, ensuring complete long-term coverage. Real-time functions, by contrast, prioritize responsiveness and efficiency, accepting intermittent data loss during failures.

To preserve long-term pipeline stability, the system enforces conservative API interaction policies including rate limiting and retry handling. This sacrifices peak throughput but protects against service bans or quota exhaustion, which could halt all downstream processes.

2.4.3. Streaming vs. Batch Harvesting

The decision to adopt batch processing over streaming was driven by platform constraints, analytical requirements, and operational considerations. Social media APIs such as Reddit's PRAW and Mastodon's REST endpoints lack native support for continuous streaming, offering only discrete query access. Attempting to emulate streaming via rapid polling would quickly violate rate limits—such as Reddit's 100 queries per minute—while adding unnecessary system complexity and resource waste.

Given the system's emphasis on trend-level analysis rather than sub-second processing, batch processing aligns better with both user needs and data characteristics. Housing discourse evolves over hours or days, making batch intervals (e.g., 15 minutes for Reddit, 5 minutes for Mastodon) sufficient for meaningful insights. In contrast, streaming architectures require persistent connections, stateful checkpointing, and advanced fault recovery—features unjustified for the expected analytical latency.

To balance scalability with simplicity, the system incorporates a hybrid model: data is collected in batches, while task distribution is handled asynchronously via Redis Streams. This enables parallel processing without adopting full-fledged streaming, maintaining operational clarity and fault isolation. The batch-first approach also supports straightforward retries and stateless failure recovery, avoiding the need for complex rollback or consistency guarantees.

The system adopts a two-phase approach: first, batch harvesting operations store raw data directly into Elasticsearch for efficient ingestion. Subsequently, a separate processing pipeline handles text cleaning, sentiment analysis, and metadata enrichment. This separation of concerns allows for independent scaling of data collection and processing resources, while maintaining the benefits of Elasticsearch's bulk indexing capabilities. Combined with the serverless Fission execution model, this architecture enables elastic resource usage—scaling during active periods and releasing resources when idle—offering a practical and sustainable alternative to continuously running streaming systems.

3. System Functionalities and Application Scenarios

3.1. Core Functionality

3.1.1. Real-time and Historical Data Collection from Reddit and Mastodon

The system continuously harvests housing-related posts and comments from both Mastodon and Reddit. The implementation is built on a Python 3.9 environment with external network access enabled, allowing for efficient API interactions with both platforms.

For Mastodon data collection, we have implemented a Fission function that maintains API cursor positions using Elasticsearch-based state management. This implementation tracks the last successful run timestamp and since_ids for each hashtag, enabling efficient incremental updates. The function includes rate limit handling with retry mechanisms, ensuring reliable data collection while respecting API constraints. The collected data, including both posts and their replies, is stored in Elasticsearch for further analysis.

The Reddit data collection is handled by a separate Fission function that collects posts and comments based on predefined parameters, with built-in error handling and retry mechanisms to ensure robust data collection. The collected data is processed and stored in Elasticsearch, maintaining a consistent data structure across both platforms.

3.1.2. Sentiment Analysis

After cleaning the content of posts and comments, we use VADER to compute sentiment metrics for each document. We choose VADER because it was specifically designed and validated for short,

informal social-media text, as documented by Hutto and Gilbert (2014) [10], achieving an F1 score of 0.96 on Twitter data and requires no task-specific training data while still providing fine-grained intensity scores. VADER integrates a lexicon of valence-rated words with syntactic heuristics, such as punctuation emphasis, capitalization cues, degree modifiers, and contrastive conjunctions, to generate four measures: positive, negative, and neutral proportions, along with a normalized compound score in $[-1, 1]$. We record each document's compound score and classify sentiment using standard thresholds: scores 0.05 are labeled "positive," scores –0.05 are labeled "negative," and scores between –0.05 and 0.05 are labeled "neutral." Both numeric scores and categorical labels are indexed in Elasticsearch for downstream analysis.

3.1.3. Interactive Visualization

Analysts work with a single Jupyter notebook that embeds both pre-built dashboards and ad-hoc plots. At the top, Kibana dashboards for overall housing discussion are shown in IFrame widgets. Below, users set a time window by editing the `T1` / `T2` variables and run code cells that fetch data from Fission HTTP routes. For example, if `T1 = 15` and `T2 = 1` the analysis shows the data from last 15 to 1 days, The notebook then displays:

- A time-series line chart of daily post and comment counts.
- Side-by-side pie charts of sentiment distribution for posts versus comments.
- Three word clouds showing the most frequent terms in posts, comments, and combined text.
- Horizontal bar charts of the top 10 bigrams and the top 10 trigrams.
- A table of five topics extracted, listing the top 10 terms per topic.
- A bar chart of the top 10 geographic entities mentioned in the text.

By modifying `T1` / `T2` and rerunning the notebook, all visualizations update automatically. Figures can be saved via `plt.savefig()` and data tables exported with `df.to_csv()` for inclusion in reports.

3.2. Application Scenarios

Our system supports several use cases to analyze housing discussion across Australia:

- **Interactive Exploratory Visualization in Jupyter**

  The client server lets users interactively explore key metrics by setting the analysis window with `T1` and `T2`. It fetches data via Fission routes and displays fundamental statistics—daily post and comment counts, sentiment breakdowns—and advanced views such as word clouds, n-gram frequency charts, topic tables and concise content summaries for the selected period.

- **Sentiment Distribution Monitoring**

  Our Jupyter Notebook retrieves sentiment counts from the `/api/all-sentiment-dist` endpoint (with `start_days=T1` and `end_days=T2`),renders a bar chart showing the proportions of positive, neutral, and negative posts and comments for the chosen interval, enabling rapid detection of shifts in public mood.

- **N-Gram Frequency and Topic Trending Analysis**

  Our Jupyter Notebook fetches cleaned text from the `/api/all-content` route, then applies Scikit-learn's `CountVectorizer` and aggregates frequency counts to display the top 10 n-grams as horizontal bar charts. For topic modeling, the notebook runs LDA on the batched term–frequency matrices to extract five latent topics, presenting each topic's top 10 terms in a neatly formatted table.

- **Content Summarization**

We also provide the content summary in client server to give users a rapid, high-level overview of conversations within the selected period.

- **Raw Data Export for External Analysis**

  The `/api/all-content` and `/api/content` endpoints allow users to download cleaned posts and comments as JSON, enabling integration with external BI tools or custom scripts.

4. Key Technical Implementations

4.1. Batch Data Ingestion Using Fission Functions

Our ingestion pipeline is implemented as two scheduled Fission functions under Kubernetes CronTriggers. The *Mastodon Harvester* queries the Mastodon API every hour, reads the last processed `since_id` from a dedicated Elasticsearch index, fetches new statuses and replies, applies sentiment scoring inline, and bulk-indexes posts and comments into separate indices using the Python Elasticsearch client's `bulk()` helper. The *Reddit Harvester* enqueues tasks in Redis Streams every 30 minutes, each task containing subreddit names and a `days_back` parameter; a consumer Fission function then asynchronously reads from Redis, fetches submissions and comment trees via PRAW, transforms them into flat JSON, and bulk-indexes them. Both functions include built-in retry loops (up to 5 attempts) and write failures to a "dead-letter" Elasticsearch index for post-mortem analysis.

4.2. Data Processing and Sentiment Analysis

After harvesting, a Fission function retrieves raw Mastodon documents from the `housing-posts` and `housing-comments` indices and Reddit documents from the `reddit-posts` and `reddit-comments` indices through `elasticsearch.helpers.scan()`.

For each record, it extracts the `content` field and applies the `clean_content()` routine, using BeautifulSoup to strip HTML tags, then regular expressions to remove hashtags, non-ASCII characters, URLs and non-word characters, collapse multiple spaces, and convert all text to lowercase. The cleaned text is then scored by VADER's `SentimentIntensityAnalyzer`, producing a compound sentiment value and categorical label. Processed documents are grouped in batches of 100 and bulk-indexed back into Elasticsearch: Mastodon data into the `posts` and `comments` indices, and Reddit data into the `posts-reddit` and `comments-reddit` indices. Any document already present in its destination index is skipped to ensure idempotence. This ETL step ensures that all stored content is both cleaned and sentiment-annotated, ready for efficient downstream querying and visualization.

4.3. API Integration and Data Harvesting

To enable following analyses and visualizations, we expose four HTTP routes in our server:

- `/api/posts-per-day` (`posts_per_day_handler`)

  Accepts integer query parameters `start_days` and `end_days`, defaulting to 7 and 0. Dates are computed in UTC and formatted as ISO strings. We issue a single Elasticsearch `search` request per index ("posts*" and "comments*") using a `date_histogram` aggregation on `created_at` with `extended_bounds` to include zero-count days. The handler returns JSON lists of {`date`, `count`} pairs for posts and comments.

- `/api/sentiment-dist` (`sentiment_dist_handler`)

Also takes `start_days/end_days`. Builds an optional `range` filter on `created_at`, then runs a `terms` aggregation on `sentiment.keyword` for "positive," "neutral," and "negative" buckets. Two separate queries, one against "posts*" and one against "comments*", are merged into a JSON object.

- `/api/all-wordcloud` (`wordcloud_handler`)

  Streams all matching documents using `elasticsearch.helpers.scan` with the same date range filter. We split each cleaned "content" string on whitespace, filter tokens by regex and a large STOPWORDS list, and accumulate counts with `collections.Counter`. The top 50 words for posts and comments are returned as JSON arrays of {`word`, `count`} objects.

- `/api/all-content` (`content_handler`)

  Also performing a `scan` over "posts*" and "comments*" with date filtering, this route extracts the cleaned "content" field, deduplicates empty strings, and returns raw text arrays for posts and comments.

4.4. Interactive Visualization and Analytics

In the client Jupyter Notebook, analysts begin by setting the `T1` and `T2` parameters to define the analysis window (in days ago) and fetch JSON data from our Fission HTTP routes listed in section 4.3 via Python's `requests` library.

At the top of the notebook, we embed Kibana dashboards through HTML iframes to provide an at-a-glance overview of discussion volume trends. Time-series data is plotted as line charts for daily post and comment counts, sentiment proportions are shown as bar charts, and term frequencies appear as word clouds using the WordCloud library.

For n-gram frequency and topic modeling, we apply Scikit-learn's `CountVectorizer` and `LatentDirichletAllocation` to the cleaned text obtained from the `/api/all-content` route; because the corpus can exceed tens of thousands of documents, we process these analyses in batches of 500 documents to avoid memory exhaustion and then aggregate the top phrase frequencies and topic term tables across batches. To generate concise summaries of large text sets, we use Hugging Face's `facebook/bart-large-cnn` pipeline, splitting any document longer than 1024 tokens into segments, summarizing each segment, and concatenating the results.

5. Data Analysis

5.1. Data Collection

Data collection is handled by two Kubernetes-scheduled Fission functions: the Mastodon Harvester and the Reddit Harvester. Each writes raw posts and comments into dedicated Elasticsearch indices for later processing. See details in Section 4.1.

5.2. Data Preprocessing

Ingested documents are retrieved from Elasticsearch, we cleaned raw data, including HTML stripping, URL and hashtag removal, converting to lowercase, and annotated with VADER sentiment scores in batches. The processed records are then bulk-indexed back for downstream queries. See details in Section 4.2.

5.3. Analysis Methods

All downstream analyses, including time-series aggregations, sentiment distributions, word-clouds, n-gram extraction, topic modeling, and summarization—consume the cleaned and indexed content using our RESTful API. See details in Section 4.3 and Section 4.4.

5.4. Results and Visualizations

5.4.1. Data Exploration

Daily Volume Trends

Figure 2 shows our system data in last 14 days. Daily new posts, shown as blue line, remained relatively steady, indicating a consistent rate of fresh discussion on housing affordability. In contrast, comment volume, shown as orange line, ranges from 800 comments at its lowest on May 24 and up to nearly 1,900 comments at its peaks on May 20. These comment spikes likely correspond to days of heightened engagement, which was perhaps driven by external events or policy developments, while the quick drop-off thereafter suggests that conversations soon returned to normal levels. Because post creation remains stable, the community's heat around affordability debates is better captured by monitoring comment volume, enabling us to identify when exactly public concern intensifies.

It is also expected that comment counts far exceed post counts in social platforms since each initial post can generate multiple replies, reflecting deeper engagement beyond the original contributions.

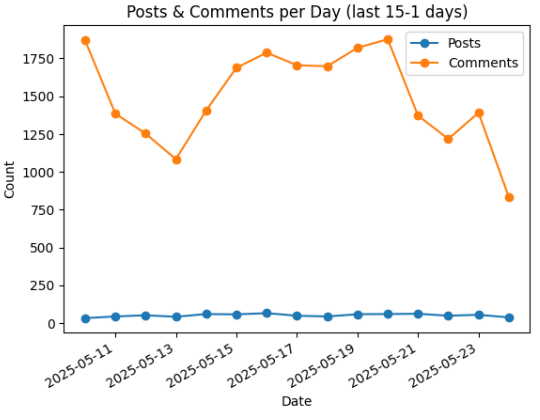


Figure 2: Posts per Day (Last 2 Weeks)

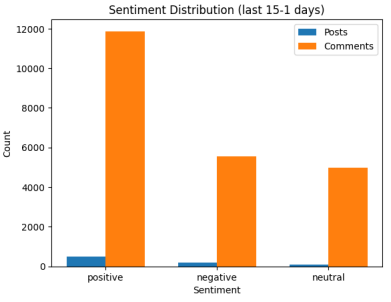
Short-Term Sentiment Statistic

In Figure 3, we analysis sentiment distribution in the last 14 days.

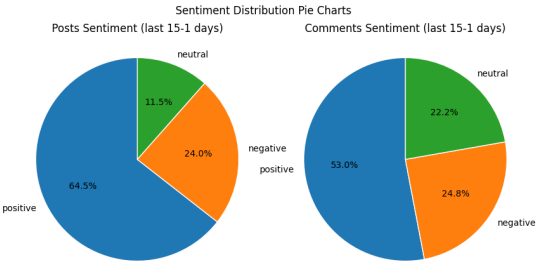
Figure 3a shows the absolute counts of each sentiment category. For posts, 497 are positive, 185 are negative, and 89 are neutral. For comments, the volumes are much higher with 11,868 positive, 5,542 negative, and 4,978 neutral. This disparity shows that while original posts are modest in number, user engagement through comments dominates the dataset and contributes a substantial share of neutral and negative responses alongside the majority of positive replies.

Figure 3b shows the relative proportions, posts are 64.46% positive, 23.99% negative, and 11.54% neutral, indicating a strong positive bias in new threads. In contrast, comments are 53.01% positive,

24.75% negative, and 22.24% neutral, revealing a more balanced mix of optimism, criticism, and ambiguous in follow-on discussions.



(a) Sentiment Distribution Bar Chart



(b) Sentiment Distribution Pie Charts

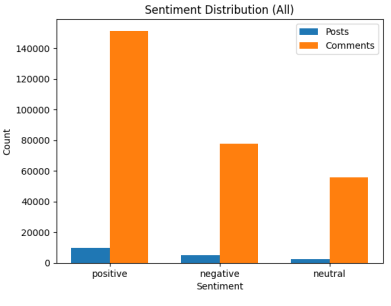
Figure 3: Sentiment Distribution (Last 2 Weeks)

Overall Sentiment Statistic

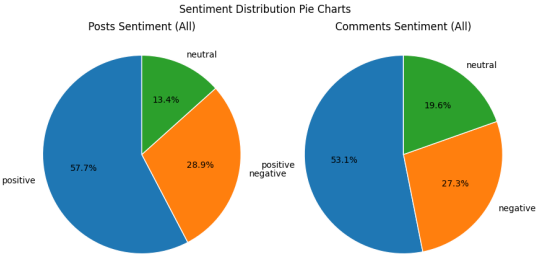
Figure 4 shows the sentiment distribution for our entire dataset to date. Overall, posts are 57.7% positive, 28.9% negative, and 13.4% neutral, while comments are 53.1% positive, 27.3% negative, and 19.6% neutral.

By comparison, over the most recent 14 days (Figure 3), posts were more overwhelmingly positive with fewer negatives and neutrals. Comments in the short term were nearly identical in positive share but showed a slightly higher neutral proportion and slightly lower negative proportion.

This comparison suggests that recent discussions have been even more positive in new posts than the long-run average while comment threads remain stable in positivity but have become marginally more neutral and less negative than historically. This indicates that short-term events can temporarily boost optimism in original posts, whereas overall engagement patterns retain a more balanced mix of sentiment. These long-term patterns align with findings by Chen et al. [5]. They shows that as online discussions mature and become more controversial, negative and neutral emotions not only emerge but persist, even as positive sentiment remains dominant.



(a) Sentiment Distribution Bar Chart



(b) Sentiment Distribution Pie Chart

Figure 4: Sentiment Distribution for All Data

5.4.2. Analysis Results

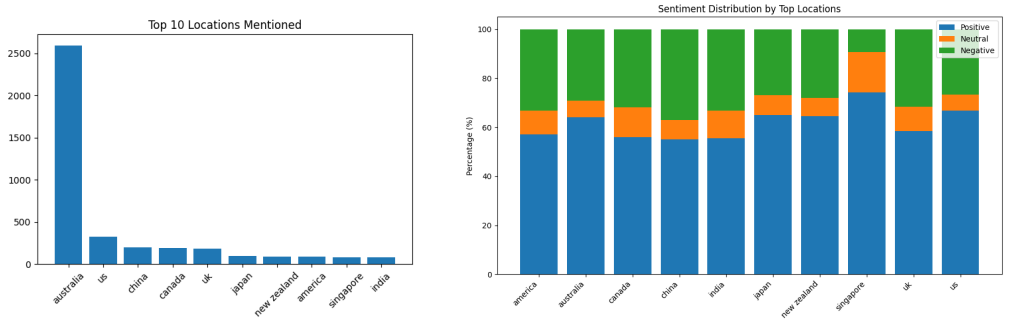
Location Analysis

Figure 5 shows the location sentiment analysis.

Figure 5a shows Australia leads mentions by a wide margin, about eight times as many as the United States and well ahead of any other location. The next most-mentioned are the US, China, Canada and the UK, then Japan, New Zealand, Singapore and India, each with fewer than 120 mentions. This skew reflects both our Australia-focused harvesting settings and the higher volume of domestic discussion around housing affordability, while still capturing a meaningful sample of international perspectives.

Figure 5b breaks down sentiment by these top locations. Most positive in sentiment are Singaporean posts with 74% positive, 17% neutral and just 9% negative sentiment. Japan and New Zealand are also very positive with around 65% with 28% negativity and 8% neutrality. Australia is also positively biased (64%), though with a larger negative proportion (29%) than the two Pacific markets. In comparison, China has the highest proportion of negative sentiment (37%), balanced by 55% positive and 8% neutral. The US, Canada, and the UK are all bunched together at 65% positive and 30% negative, with 10% neutrality. India is in the middle of this group (56% positive, 32% negative, 12% neutral). These trends indicate that although Australia accounts for most of the discussion, optimism is most intense in smaller but highly active markets such as Singapore, Japan and New Zealand.

These charts show that while housing affordability conversations are predominantly positive across all regions, there is clear geographic variation: some markets (e.g. Singapore, Japan) express stronger optimism, whereas others (e.g. China, Canada, the UK) sustain a more critical or ambivalent tone. As demonstrated by Zhan et al. (2019) [17], neighborhoods with higher rental costs or poorer infrastructure exhibit a greater share of negative sentiment. Consistent with their findings, our analysis finds that Canada and the UK, where markets known for high housing costs, whereas Singapore and Japan rank highest in positive sentiment. This alignment confirms that social media sentiment analysis can reliably capture geographic variations in housing satisfaction across different regions. Tracking these differences helps policy recommendations to regional sentiment climates.



(a) Top Mentioned Locations

(b) Sentiment Distribution by Top Locations

Figure 5: Locations Sentiment Distribution Analysis

Lexical Trends

Figure 6 illustrate the principal vocabulary of housing discussions of the last two weeks. In initial postings, "property", "house", "people", "years" and "time" predominate, suggesting that authors frame housing most conspicuously as a long-term investment and life milestone. Frequent deployment

of economic language such as "mortgage", "income", "rent" and "deposit" foregrounds paramount concerns with affordability and how home ownership is to be funded. The saliency of "super" and "pay" also works to underscore connections between retirement savings and home expenses.

Comment threads prioritize lived experience, with "people", "house", "money" and "years" most salient again. Terms such as "work", "buy" and "tax" are more visibly utilized in comments compared to posts, reflecting practical arguments regarding earning power, consumption choices and policy effects. When both the posts and comments are merged, "people", "house", "time", "money" and "work" are the biggest tokens, illustrating how personal finance, work and time horizons influence the complete discussion. On the whole, these lexical trends reveal that both authors and commenters revolve around financial limitation and Australia's future horizon of housing

Figure 6: Word Cloud (Last 2 Weeks)

Term Frequency Analysis

In Figure 7, we compare raw counts of the top twenty terms. "people" leads overall usage at over 3 200 occurrences, driven largely by comments. "House" follows with more than 2, 400 mentions, then "years," "time," "work" and "money," each appearing between 1, 500 and 1, 900 times. Original posts contribute a smaller but still meaningful share of each term. Most notably "house" (about 300), "people" (about 180), and "property" (about 230). Words such as "pay," "housing," "income," "year" and "mortgage" appear between 120–240 times in posts and 1, 000–1, 700 times in comments. This distribution confirms that while comments dominate in volume, authors consistently introduce core concepts, including housing, finance and temporal outlook, that drive the conversation.

Top 10 Bigrams

Figure 8 shows "Affordable housing" appears roughly 650 times and "housing crisis" about 550 times, highlighting that both supply shortages and cost pressures dominate the conversation. "Social housing" (around 320), "real estate" (around 260), "housing market" (around 250) and "public housing" (around 190) each exceed 150 mentions, reflecting a focus on both private and government-supported solutions. Lower-frequency but still significant bigrams include "years ago" (around 180), "capitalism politics" (around 170) and "housing homelessness" (around 170), which point to historical framing and broader socio-economic critique. The prevalence of these two-word patterns indicates that participants view housing affordability not only as an urgent economic challenge but also as a long-standing structural problem.

Topic Analysis

Table 1 summarizes five LDA-derived topics:

– Topic 1: Purchase Timeline and Financing

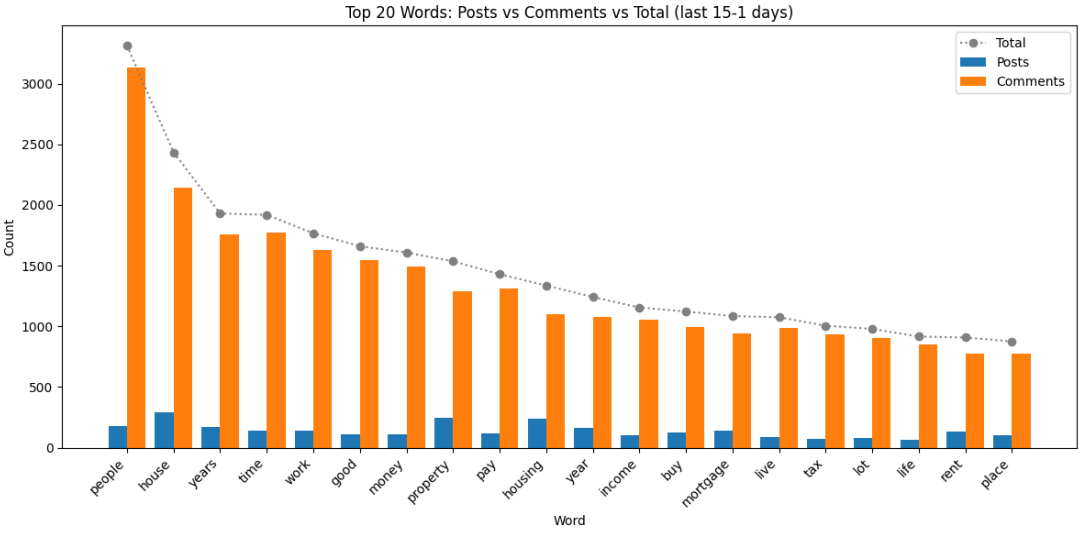


Figure 7: Top 20 Words (Last 2 Weeks)

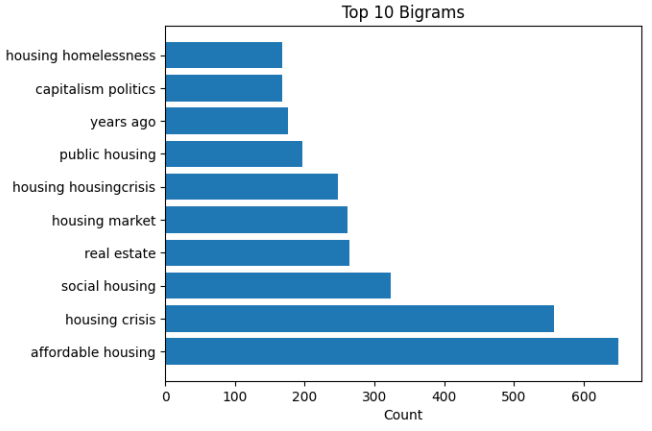


Figure 8: Top 10 Bigrams

Terms like "years ago," "debt recycling," "buy house," "mortgage broker," and "tax free" indicate discussions around how long it takes to save and strategies for mortgage financing and debt management.

– Topic 2: System Messages and Noise
  Words such as "performed automatically," "contact moderators," "message compose," and "com au" suggest this topic captures harvesting or platform-related noise, including automated notifications and moderation prompts.
– Topic 3: Socio-Demographic Concerns
  Keywords like "middle class," "young people," "mental health," and "house prices," point to debates about how housing affects different social groups, especially younger generations and their mental well-being.
– Topic 4: Investment and Long-Term Strategy

Phrases such as "real estate," "negative gearing," "long term," "20 years," and "aged care" reveal a focus on property investment strategies, tax incentives, and planning horizons for homeownership.

– Topic 5: Public and Shared Housing
Terms like "social housing," "public housing," and "share house" highlight conversations about alternative housing models, government-provided options, and co-habitation arrangements.

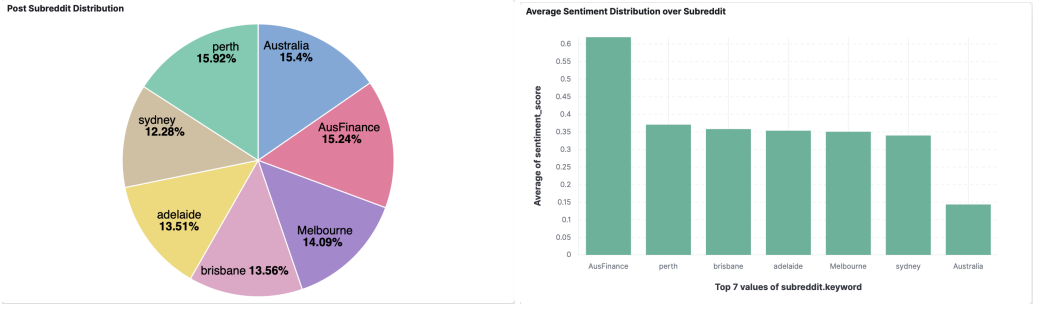
Except for Topic 2, which clearly represents system noise from the data-harvesting process, the other four topics are highly relevant to our project's focus on housing affordability, encompassing personal financial planning, socio-demographic concerns, investment strategies, and public policy options.

Table 1: LDA Topics and Top 10 Terms

Topic 1	**Topic 2**	**Topic 3**	**Topic 4**	**Topic 5**
years ago	don want	middle class	real estate	don know
debt recycling	good luck	young people	sounds like	social housing
buy house	com au	mental health	don know	don think
don know	financial advice	don want	long term	public housing
sounds like	house prices	looks like	don need	share house
mortgage broker	performed automatically	house prices	years ago	don want
tax free	message compose	new zealand	aged care	sounds like
don think	contact moderators	doesn mean	negative gearing	make sure
long term	moderators subreddit message compose	want live	make sure	years ago
ve got	action performed automatically	feel like	20 years	ve got

Subreddit Analysis

Figure 9 shows that posts are distributed almost evenly across seven subreddits. Numberically, *Perth* is 15.92%, *Australia* is 15.4%, *AusFinance* is 15.24%, *Melbourne* is 14.1%, *Brisbane* is 13.6%, *Adelaide* is 13.5% and *Sydney* is 12.28%. However, sentiment varies widely: *AusFinance* is most positive (avg. score ≈ 0.62), while the broad "Australia" subreddit is least (≈ 0.14). *Perth*, *Brisbane*, *Adelaide* and *Melbourne* cluster around 0.34, reflecting moderate optimism, and *Sydney* sits near 0.34. This pattern suggests that discussions in finance-oriented spaces reflect the highest confidence in housing affordability, whereas more general or broad-interest forums exhibit greater ambivalence.



(a) Subreddit Distribution on Posts (b) Subreddit Sentiment Distribution

Figure 9: Subreddit Sentiment Distribution Analysis

Summarization of posts and comments
The following automated summaries reveal how discussion focus shifts with aggregation period:

[Daily Summary]

Capitalists will always attempt to turn returns from into returns from i think and both missed this if a capitalist can artificially constrain supply or make demand non optional then they are no longer engaged in a classical or neo classical not even mode of production they are engaged in extraction. tech investors want companies to lose money and grow as quickly as possible is that they want to 1 push any competition out of the market 2 get their users hooked on their product to the extent that those two things are accomplished

[Weekly Summary]

The government will spend 21 1 million to support 2 500 additional households struggling to pay for rental. It would take 35 years in toronto 49 in vancouver and more than 20 in hamilton and waterloo just to return to early 2000s affordability levels.

[Monthly Summary]

Housing crisis needs policy attention but too many of us may be living in houses that no longer suit our current needs as moving is too difficult. It would take 35 years in toronto 49 in vancouver and more than 20 in hamilton and waterloo just to return to early 2000s affordability levels archdaily tr voux 84 housing units tectoniques corporations that use rental properties as a way to generate returns for investors such as private equity firms.

These summaries demonstrate that as the aggregation window expands from daily to weekly to monthly, the discussion focus shifts accordingly. Specifically, from ideological critiques of market actors in the daily summaries, to concrete reporting on policy interventions in the weekly summaries, to broader human-centric reflections and corporate accountability in the monthly summaries, highlighting the value of multi-scale summarization for capturing both the emotional undercurrent and the practical realities of the housing affordability debate.

6. System Robustness

6.1. Fault Tolerance

6.1.1. Fault Tolerance Overview

The system adopts a multi-layered fault tolerance strategy to ensure continuous operation under infrastructure or application-level failures. Kubernetes orchestrates data ingestion jobs and Fission functions, automatically restarting failed pods using the `restartPolicy: OnFailure` directive and retrying failed jobs according to the specified `backoffLimit`. Sensitive configurations and credentials are externalized via Kubernetes `ConfigMaps`, enabling stateless re-provisioning without manual intervention.

6.1.2. Retry mechanisms in harvester scripts

The harvester components implement structured retry mechanisms with exponential backoff to handle transient API failures. For Reddit, all API interactions are encapsulated within a helper function that detects `HTTP 429 Too Many Requests` errors, applies delays based on the `Retry-After` header, and retries up to five times. For other exceptions, an exponential backoff schedule is used.

The Mastodon harvester applies a similar retry strategy, parsing the `X-RateLimit-Reset` timestamp and injecting jitter to prevent synchronized retries across instances. Retry attempts are capped to five iterations per request.

In addition to API-level retries, the system enforces task-level fault tolerance through Redis Streams. Each harvesting task includes retry metadata, such as `retry_count` and `max_retries`, enabling automatic requeuing of failed tasks until retry limits are exceeded.

6.1.3. Stateless Fission functions for recovery

The serverless data processing components are implemented as stateless Fission functions. These functions reinitialize required clients (e.g., Elasticsearch) upon each invocation and do not retain in-memory state across executions. This stateless design allows failed or evicted function pods to be immediately replaced by Kubernetes without requiring state reconciliation.

Configuration and environment variables are injected via Kubernetes `ConfigMaps`, ensuring consistent behaviour across all function instances. Combined with externalized state management in Elasticsearch and Redis, the system ensures rapid and seamless recovery from unexpected disruptions.

6.1.4. Avoidance of single points of failure via pod replication

To eliminate single points of failure, the system relies on Kubernetes-native redundancy and replication strategies. Each harvesting job is deployed as an independent `Job` or `CronJob`, configured with `backoffLimit: 3` and `restartPolicy: OnFailure`. Failed pods are automatically rescheduled on healthy nodes without human intervention.

Fission function pods are managed by the Fission controller and replicated across multiple nodes, enabling continuous availability even in the presence of node-level failures. The Elasticsearch cluster is provisioned as a multi-node `StatefulSet` (or managed via an operator such as ECK), ensuring that the failure of a single node does not compromise index accessibility.

Task distribution through Redis Streams supports concurrent processing via consumer groups. Failed consumers are detected automatically, and their pending tasks are reassigned to active instances after 60 seconds of inactivity, ensuring uninterrupted task processing and load balancing.

6.2. Scalability

6.2.1. Scalable Data Ingestion via Kubernetes Jobs

Historical data collection, particularly from Mastodon, is executed through multiple Kubernetes `Job` definitions, each targeting a specific hashtag. These jobs are configured to run concurrently, allowing efficient parallel harvesting as cluster resources permit. For Reddit, a `CronJob` is responsible for triggering the task producer every 15 minutes. This mechanism ensures consistent workload generation and allows the system to respond to increased posting volume—e.g., during high-activity hours—by scheduling overlapping job pods.

The degree of parallelism is fully configurable. By adjusting the `parallelism` parameter in each job specification and configuring the `concurrencyPolicy` in `CronJob` resources, the system can launch multiple harvesting pods in parallel to process different keyword-subject combinations independently.

6.2.2. Elastic Task Processing via Redis Streams

To decouple task generation and execution, the Reddit harvesting pipeline employs Redis Streams as a message queue. Tasks generated by the producer are stored in the `reddit_harvest_stream`, which supports consumer groups. This enables multiple harvester pods to operate as independent consumers, processing tasks in parallel. If the backlog in the stream grows—indicating increased demand—additional consumer pods can be spun up to drain the queue more efficiently, preserving near real-time performance.

6.2.3. Autoscaling API and Function Workloads

At the API and processing layers, Fission serverless functions provide elastic scaling. These stateless functions are managed by the Fission controller, which can automatically spawn additional pods in response to increased request volume. Functions reinitialize with each invocation, loading configuration from Kubernetes `ConfigMaps` and relying on external systems such as Elasticsearch or Redis for state persistence, allowing them to scale rapidly without synchronization overhead.

For non-Fission workloads—such as a potential custom API gateway—Kubernetes Horizontal Pod Autoscalers (HPAs) may be applied to dynamically adjust the number of replicas based on CPU utilization or application-specific metrics like request latency.

6.2.4. Scalable Storage via Elasticsearch

The platform uses Elasticsearch as its primary data store. Although the deployment YAML files do not specify a full multi-node configuration, the system is designed to connect to a Kubernetes-managed Elasticsearch cluster. In production, this would typically be provisioned via a `StatefulSet` or an operator such as ECK.

Elasticsearch scales horizontally through sharding, distributing index data and query load across multiple nodes. As data volume increases, new nodes can be added to the cluster, and existing indices can be rebalanced to maintain performance. Replica shards further enhance read throughput and fault tolerance, enabling multiple concurrent read operations without overloading individual nodes.

6.2.5. Infrastructure-Level Resource Scaling

Kubernetes orchestration underpins the system's overall elasticity. Pods for jobs, harvesters, APIs, and functions are all containerized, making replication straightforward. Kubernetes supports resource limits and requests to ensure fair allocation, and can scale deployments based on resource availability. When integrated with a Cluster Autoscaler or deployed in a cloud platform like NeCTAR, additional worker nodes can be provisioned automatically as resource demands grow.

7. Discussion

7.1. Pros and Cons of System Architecture

7.1.1. NeCTAR Research Cloud

The NeCTAR Research Cloud, built on OpenStack, provided a flexible infrastructure to support our cloud-native project. It help us to deploy and scale virtual machines, attach volumes, store persistent data, and operate in a customizable environment suitable for research applications.

One of the major platform's strengths is its scalability. It eliminates the need for dedicated physical servers by allowing virtual instances to be spun up or down based on demand. This was particularly beneficial for our data-intensive system, where computational requirements fluctuated depending on the size of the datasets being processed. NeCTAR supports both horizontal scaling and vertical scaling. Practically, it allow us to deploy additional instances or upgrade instance specifications.

In terms of performance and efficiency, NeCTAR's infrastructure is physically hosted in University of Melbourne data centers. This geographic proximity ensured low-latency communication between instances and efficient intra-cloud data transfer[13].

However, we encountered several limitations. One significant challenge was Quota limitations. The number of vCPUs, memory, and disk space allocated per project was limited, which restricted the scale of our experiments and required frequent resource recycling.

Additionally, we faced system availability issues during critical operations. In one case, the NeCTAR dashboard became unresponsive while deleting a Kubernetes cluster, leaving it in a "delete in progress" state for several days and blocking further progress. (See more detail in Section 7.2.1)

Moreover, because NeCTAR is a remote infrastructure, all data transmission depends on the public internet. This introduces potential security concerns, especially when handling large-scale or sensitive datasets. Encryption and secure access control mechanisms must be configured manually. Combined with occasional dashboard instability during instance or volume operations, these factors required additional time and care during deployment[14].

7.1.2. Kubernetes

We adopt Kubernetes as a container orchestration. One of its greatest strengths is continuously automated resource management. It dynamically adjusts workloads in response to CPU or GPU utilization, which is especially useful for fluctuating AI workloads or distributed computing scenarios.

The platform supports efficient load balancing and service discovery, ensuring stable performance even under heavy traffic. Built-in networking and security features further strengthen multi-tenant isolation and secure cluster operation[16].

However, Kubernetes also presents several limitations. The platform has a steep learning curve, especially for newcomers unfamiliar with containerization and cluster architecture. Configuring components such as services, volumes, and ingress controllers can be complex and error-prone. Debugging issues is often nontrivial due to the distributed nature of the system; failures may originate from pods, networking layers, storage volumes, or DNS resolution[12]. Additionally, Kubernetes consumes significant resources even in lightweight setups like k3s, which can be challenging when operating under strict VM quotas such as those provided by NeCTAR. Cold start delays and cluster scheduling latency may also affect performance in time-sensitive applications.

7.1.3. Fission

Fission is a serverless framework built on Kubernetes that made it easy for us to deploy modular Python functions with minimal overhead. While it supports out-of-the-box environments for languages like Python and Node.js, it also allows developers customize environments to support specific dependencies. This flexibility made it possible to integrate advanced NLP packages, although it did require us to build a custom Docker image due to the complexity of its dependencies.

Fission also offers fine-grained control over infrastructure, allowing us to specify resource constraints such as minimum and maximum CPU/memory, scale range, and pod-level configurations. These settings enabled us to balance resource usage within NeCTAR's limited quotas while maintaining function responsiveness[3].

However, observability and debugging remained limited. Fission provides minimal built-in tools for log aggregation or function tracing, which made troubleshooting errors difficult, especially when functions failed silently. In our deployment, log access was fragmented across pod logs, making it harder to diagnose issues without direct access to the underlying Kubernetes cluster.

In addition, when a function is triggered after a period of inactivity, Fission needs to allocate resources, start the necessary pod, and load the function environment. This initialization can introduce noticeable delays—particularly for larger environments or resource-constrained systems like those running on NeCTAR.

7.1.4. ElasticSearch

Elasticsearch in our system serves as the core database for managing social media posts and comments, offering near real-time indexing and retrieval capabilities. One of the key advantages of Elasticsearch is its powerful full-text search support and its flexible query DSL, which allowed us to filter and aggregate data by sentiment, topic, and location with ease. This enabled advanced query use cases such as retrieving comments by state, ranking posts by sentiment score, or analyzing topic frequency over time.

A major strength of Elasticsearch is its seamless integration with Kibana, a web-based data visualization and dashboard tool. Kibana allowed us to quickly explore indexed data, monitor ingestion patterns,

and visualize sentiment and topic distributions through interactive charts and time-series plots—all without writing frontend code. This significantly accelerated our ability to validate processing results and share insights with non-technical stakeholders[7].

Elasticsearch's schema-less nature and JSON-native structure also made it easy to ingest enriched documents, such as those annotated with NLP outputs like VADER sentiment scores and BERTopic labels. Its support for aggregations and filters enabled efficient data slicing without relying on complex joins. However, Elasticsearch also comes with limitations. It is relatively memory-intensive, and improper resource allocation—particularly under constrained cloud environments like NeCTAR—can result in slow queries or index failures.

Additionally, misconfigured field mappings may lead to unexpected behavior or query mismatches, and debugging such issues can be unintuitive. Elasticsearch also lacks native support for relational joins, which required us to denormalize post and comment data for indexing[8].

7.2. Issue and Challenges

7.2.1. Cloud Issue

Early in the project cycle, the task of deploying of a stable Kubernetes cluster on the NeCTAR cloud infrastructure was fraught with issues. Since the successful deployment of the cluster nodes is fundamental to the entire basis of the cloud analytics project, issues in this category of challenges have significantly delayed our progress and limited project scope.

Initially, commands were issued to the NeCTAR through the Openstack client api to create clusters from the provided University of Melbourne Kubernetes template. However, although the commands were received by NeCTAR, and a cluster group was being constructed, the cluster never progressed to a healthy state. Assuming it is due to high demand on the cloud's resources, we unfortunately elected to leave the cluster creation process alone in hopes that it would eventually resolve itself.

When the cluster continued to fail to progress to a healthy stable state, we then attempted to forcefully delete the cluster through the NeCTAR web dashboard, and attempt create the cluster again. This then, in fact, caused more issues to occur, as the cluster was now stuck in a permament state of 'delete-in-progress' instead. In addition, newer attempts to create clusters in our NeCTAR project was met with subsequent failures to progress as well.

It was at this point that we contacted NeCTAR and the teaching staff for support, as although minor progress was made in scripting parts of the functionalities, such as harvesting and preprocessing, testing locally, the absence of a cluster infrastructure severely hindered our understanding of how all of these components could connect with each other, and debugging any communication errors that may occur in between.

In addition, due to the delayed setup of the cloud infrastructure, our local scripting and testing of data harvesting from the publicly available reddit and mastodon APIs were initially performed under very different assumptions on how the script would be ran, which led to further challenges as detailed in the next section.

7.2.2. Harvest Challenge

In gathering historical Reddit data to better visualise housing sentiment over time, we ran into one of hard limitations of the Reddit API (PRAW), where it can retrieve 1000 posts within a specific search ordering (e.g. Hot, New, Top). This meant that it is essentially impossible to gather an accurate and complete Reddit dataset, as the API did not provide a reliable mean to broaden the search results beyond the most relevant 1000 posts of any given query.

In addition, this gathering of historical data was originally tested locally due to cloud issues, and when moved to fission, gathering 1000 Reddit posts annd also retrieving their comments, whilst operating within the API rate limit, resulted in guaranteed timeout by fission, as the time taken is simply too long.

And so historical Reddit data harvesting methods was changed dramatically after the cluster was eventually stable, which required a substantial amount of time. To remedy this issue, we have also looked into uploading existing Reddit data dumps, specifically the pushift dumps which included data ranging from 2005 to 2024. We elected not to upload all relevant data to Elasticsearch as the storage and computation needed would be beyond the scope for our project.

It is important to note that this challenge does not affect our data harvesting methods for current/new posts that are submitted on the our chosen Australia-related subreddits such as r/AusFinance, as this is performed through a repeated scheduling of a small fission function that only gather small amount of posts each time, similar to the Mastodon harvester.

Another challenge that data gathering presented was that posts from both sources, Reddit and Mastodon, lacked geolocation metadata. Since we longer have free access to the Twitter/X API, extracting meaningful geographic-based information on each post is mainly inferred through the search query submitted to the API when harvesting.

For example, when harvesting data from the subreddit r/melbourne specifically, we can infer the geographic location of posts and comments are, in fact, from Melbourne, or at the very least, most closely related to Melbourne. This inferring of location is also similarly performed for Mastodon data, where an Australian state name is provided as a secondary tag when querying the API for data.

To ensure relevance to housing, we made sure to only ingest data that contains certain housing related keywords to ElasticSearch.

Furthermore, the posts are often emotionally charged, and highly contextual, and so sentiment analysis can often be inaccurate. To mitigate this effect then, we made sure to preprocess, and filter any 'bad' (i.e. short, or empty) responses from the public API endpoint, to give the vaderSentiment Analyser the best chances for accurate results.

7.2.3. Processing Reindex Challenge

When we attempted to backfill over 200,000 historical Reddit comments by invoking our reindexing logic directly from a Fission HTTP route, any scan of more than roughly 75,000 documents invariably ran longer than the 60 s route timeout:

```
http error [error executing HTTP request:
 Get "http://127.0.0.1:52927/fission-function/analysis-reddit-comments":
 function request timeout (60000000000)s exceeded]
```

This failure occurred because the combination of establishing a long-lived Elasticsearch scroll cursor, fetching large batches, and performing inline sentiment analysis for each document simply could not be completed within the synchronous HTTP invocation window.

The mitigation strategies we tried each fell short:

- **Smaller bulk sizes**

  Reducing the batch from 1,000 to 100 documents lowered the time per bulk write, but dramatically increased the number of ES calls and scroll iterations, so the total scan plus analysis time remained above the timeout threshold.

- **Extended route timeout**

  Although Fission allows increasing the execution deadline, setting it significantly above 60 s risked blocking container resources for prolonged periods and did not address the root cause—the synchronous HTTP model itself.

- **Threaded background job**

Spawning the reindex task in a daemon thread freed the HTTP handler to return immediately, but the thread still ran inside the Fission pod with the same CPU/memory constraints and could be terminated by Kubernetes if resource limits were hit

Only by moving the job to a Kubernetes CronTrigger, running as a dedicated, long-lived Fission function outside of an HTTP request context, were we able to process the entire index reliably. In that mode, the function pod persists across invocations, can be given higher resource limits, and is not subject to the HTTP route timeout, ensuring idempotent, off-peak backfilling of all 200,000+ documents.

7.2.4. Other Challenges

We have also ran into issues in the process of packaging code into archives and sending to fission. Since all of our team members use operating systems other than Linux, we all ran into minor yet frustrating compatibility issues with file formats.

For example one of our team members experienced the following error

```
(base) linyingwei@tiantianshuidajue history-mastodon
% kubectl logs -f job/history-mastodon-housing | tail -n 20
Error from server (BadRequest): container "history-mastodon-housing" in
pod "history-mastodon-housing-66tfq" is waiting to start:
trying and failing to pull image
```

This is due to the image built on their MAC system was not suited for the x86 system on the cluster, and required that the image be recreated but targetted specifcally for the linux/amd64 platform.

Another team member also ran into a problem with fission not recognising files within the zip archive. The build.sh shell script which controls the python package environment simply was missing for fission, even though the file is within the zip archive when unpacked. Eventually a workaround was found, as they chose to duplicate a functioning zip archive with a build.sh file, replace the python script instead and upload that to fission, concluding that perhaps the zip command in WSL functions differently than Linux systems, and a 'bad' zip file was produced.

8. Conclusion

8.1. System Conclusion

We developed a simple, cloud-native pipeline that automatically collects data with Fission functions on Kubernetes, indexes it in Elasticsearch for near-real-time search, exposes a RESTful API for queries, and offers interactive dashboards in Jupyter and Kibana. Running on NeCTAR Research Cloud, it continuously ingests, cleans, and analyzes high-volume Reddit and Mastodon discussions with minimal overhead. Automated retries, modular function design, and horizontal autoscaling ensure the system stays fast and reliable even under tight CPU, memory, and storage limits.

8.2. Analysis Conclusion

In this project, we applied various analytical techniques, including time-series volume trends, sentiment and NLP analysis, such as word clouds and bigrams, LDA topic modeling, and automated multi-scale summarization.

These analyses give us a clear, comprehensive view of social-media discussions on housing. First, word clouds and bigram counts shows that conversations focus on affordability, supply, and broader socio-economic issues. Moreover, LDA topic modeling uncovers five key themes, including personal finance, socio-demographic concerns, investment strategies, and public or shared housing. Finally, automated summaries at daily, weekly, and monthly scales show how the debate shifts from ideological

critique, to policy discussion, to personal stories. To sum up, these insights demonstrate that social-media sentiment and topic patterns closely follow real-world changes in housing affordability, providing timely, detailed context that complements traditional surveys and economic indicators.

8.3. Future Direction

For system improvements, we will integrate centralized log aggregation and distributed tracing into a single platform. This unified observability will allow us to detect failures faster, pinpoint performance bottlenecks, and reduce mean time to resolution. We will also automate our CI/CD pipelines with GitHub Actions to ensure consistent, repeatable deployments and shorter update cycles, which in turn minimize manual errors and downtime.

On the analysis side, we will enhance text preprocessing by expanding custom stopword lists to remove platform-specific noise and applying finer-grained regex filters to improve data quality. We will keep our LDA model with optimized hyperparameters and monitor coherence scores to achieve clearer, more actionable topics. Finally, by ingesting hedonic rental price models as auxiliary features, we can contextualize sentiment trends against real market fluctuations—making our insights more relevant to policy and economic analysis. This roadmap supports iterative enhancements driven by user feedback and observed system metrics.

References

[1] O Abu Oun and Tamas Kiss, *Job-queuing and auto-scaling in container-based cloud environments*, 10th International Workshop on Science Gateways, IWSG 2018, CEUR Workshop Proceedings, 2019.

[2] Abdulazeez Abdulazeez Adeshina, *Building python web apis with fastapi: A fast-paced guide to building high-performance, robust web apis with very little boilerplate code*, Packt Publishing Ltd, 2022.

[3] Yasmina Bouizem, *Fault tolerance in faas environments*, Ph.D. thesis, Université Rennes 1, 2022.

[4] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes, *Borg, omega, and kubernetes*, Communications of the ACM **59** (2016), no. 5, 50–57.

[5] Kai Chen, Zihao He, Rong-Ching Chang, Jonathan May, and Kristina Lerman, *Anger breeds controversy: Analyzing controversy and emotions on reddit*, arXiv preprint arXiv:2212.00339 (2022).

[6] Štěpán Davidovi and Kavita Guliani, *Reliable cron across the planet: ... or how i stopped worrying and learned to love time*, Queue **13** (2015), no. 3, 30–39.

[7] Bharvi Dixit, *Elasticsearch essentials*, Packt Publishing Ltd, 2016.

[8] Clinton Gormley and Zachary Tong, *Elasticsearch: the definitive guide: a distributed real-time search and analytics engine*, " O'Reilly Media, Inc.", 2015.

[9] Lin Hu, Shenjing He, Zixuan Han, He Xiao, Shiliang Su, Min Weng, and Zhongliang Cai, *Monitoring housing rental prices based on social media: An integrated approach of machine-learning algorithms and hedonic modeling to inform equitable housing policies*, Land Use Policy **82** (2019), 657–673.

[10] C. J. Hutto and Eric Gilbert, *Vader: A parsimonious rule-based model for sentiment analysis of social media text*, Proceedings of the Eighth International AAAI Conference on Weblogs and Social Media (ICWSM-14), 2014, pp. 216–225.

[11] Nikita Kathare, O Vinati Reddy, and Vishalakshi Prabhu, *A comprehensive study of elasticsearch*, International journal of science and research (IJSR) (2020).

[12] Paridhika Kayal, *Kubernetes in fog computing: Feasibility demonstration, limitations and improvement scope*, 2020 IEEE 6th World Forum on Internet of Things (WF-IoT), IEEE, 2020, pp. 1–6.

[13] Zheng Li, Rajiv Ranjan, Liam O'Brien, He Zhang, Muhammad Ali Babar, Albert Y Zomaya, and Lizhe Wang, *On the communication variability analysis of the nectar research cloud system*, IEEE Systems Journal **12** (2016), no. 2, 1506–1517.

[14] Bernard F Meade and Christopher J Fluke, *Evaluating virtual hosted desktops for graphics-intensive astronomy*, Astronomy and computing **23** (2018), 124–140.

[15] Sebastian Tallberg, *A comparison of data ingestion platforms in real-time stream processing pipelines*, (2020).

[16] Indrani Vasireddy, Prathima Kandi, and S Gandu, *Efficient resource utilization in kubernetes: A review of load balancing solutions*, International Journal of Innovative Research in Engineering & Management **10** (2023), no. 6, 44–48.

[17] Fang-Bin Zhan, Tom Knudson, and Andy Spain, *Sentiment-based spatial analysis of residential satisfaction from twitter data*, Journal of Housing and the Built Environment **34** (2019), no. 3, 877–898.