Tweet Disaster Analysis

Project Report

CS 4485.0W1 Team 8

Allison Nguyen, Shraddha Patel, Michelle Sang,

Anjali Singh, Hephzibah Sujoe

Chapter 1: Project Overview and Scope

Business Context

I. Business Problem and Objective

In times of crisis, getting accurate and timely information can be challenging. While social media platforms like BlueSky provide real-time updates, the overwhelming volume of posts makes it difficult to identify critical information about emerging disasters and their severity.

Traditional news sources often lack the personal experiences and on-the-ground perspectives that people share online. While official reports provide broad overviews, they miss the realities of individuals directly affected by disasters. These personal narratives and timely observations provide crucial context about specific local conditions and the human impact of unfolding events.

Our project addresses this need by analyzing BlueSky data to detect and map emerging crises. Our platform filters through the noise to present users with a clear, visual representation of where disasters are occurring and their severity levels. With this information at their fingertips, users can stay informed about developing situations in their area or regions of interest, make better safety decisions, and understand the potential impact of events as they unfold.

Our platform allows the public to submit help requests that are displayed to first responders with actionable information. Community members can indicate their specific needs during emergencies, which our system then organizes for emergency services to view. This direct connection ensures that first responders receive timely, location-specific details about where help is needed and what resources are required, enabling more targeted assistance and faster response times to those most affected.

This accessible tool puts critical disaster intelligence directly in users hands, helping them stay safe and informed during uncertain times, while providing emergency personnel with actionable insights to enhance their response capabilities.

II. Key Project Risks

During the implementation of this project, several risks were identified and considered that could potentially impact the timeline, performance, and accuracy of our product:

A. **System Integration Risk:** Managing the integrations between our backend, frontend, database, and deployment environment posed a significant risk. Ensuring compatibility across all components required careful planning and frequent testing to avoid failures during runtime.

B. **Computational Resource Constraints:** Running large language models (LLMs) demanded substantial computational power. However, our available hardware was significantly below the recommended specifications, resulting in noticeable slower processing times. Overall, this affected the efficiency of our model deployment and testing.

C. **Budgetary Limitations:** The project encountered financial constraints, especially regarding cloud hosting services and API usage. Our initial choice of tools and platforms were too expensive, leading us to adopt more affordable alternatives. While cost-effective, these alternatives may not offer the same level of performance or reliability as our originally preferred options.

D. **Data Quality and Noise in Input Sources:** A high volume of spam or non-crisis related tweets significantly affected the system's response efficiency. This noise in the data required additional filtering and classification efforts to maintain accuracy

E. **Risk of AI-Generated Inaccuracies:** Due to our implementation with large language models (LLMs), there was a risk of generating inaccurate or misleading information. This posed significant challenges in maintaining the system's credibility and reliability, particularly in high-stakes crisis scenarios where timely and precise information is critical. To address this, we included a disclaimer informing users that the information may not always be accurate; however, this precaution also impacted the perceived trustworthiness of our platform.

Project Scope

This project aims to build a timely disaster analytics platform that leverages social media data to provide timely, structured, and geographically relevant crisis information. By collecting, processing, and analyzing posts from the BlueSky API, the platform extracts meaningful insights such as disaster type, severity, location, and sentiment.

I. Scope of Usage and Core Functionality

A. **Disaster Data Collection**: Automated tweet retrieval using a keyword-based querying system that tracks 15+ disaster types (e.g., Flood, Fire, Earthquake, Hurricane).

B. **Data Preprocessing**: Cleaning and structuring of tweets through text normalization, language filtering, and metadata extraction.

C. **Disaster Classification and Labeling**: A fine-tuned LLM pipeline that determines whether a tweet reflects a genuine disaster, identifies the disaster type and location, and calculates a severity score.

D. **Analytics Dashboard**: A React-based front end that displays trends using interactive charts, KPIs, and visual filters including disaster type, date range, severity threshold, and

location.

E. **Top 10 Tweet Table**: Interactive map that visualizes the top 10 disaster-related tweets globally using geolocation.

F. **Recent Watch Feed**: Updated feed showing the most recent filtered disaster tweets for quick situational awareness.

G. **Disaster Risk Heatmap**: Heat-based geographical visualization that shows disaster-prone areas with severity-based coloring.

H. **Emergency Assistance Request**: Interface for users in affected areas to request aid and submit distress messages.

I. **First Responder Login**: Secure login for authenticated responders to view requests, dispatch help, and prioritize urgent zones.

J. **Advanced Filtering**: All visualizations dynamically update based on selected filters. Disaster type normalization ensures consistency across data.

K. **Location Intelligence**: Extracted location names are geocoded to latitude/longitude, allowing regional severity analysis and mapping.

L. **Cloud Deployment**: The backend pipeline is containerized using Docker and deployed manually on a shared AlmaLinux VM. Cron jobs run the data collection and labeling scripts daily to keep the database up to date.

II. Developer Information

A. **Allison Nguyen:** Team Lead and Scrum Master that coordinated closely with supervisors to align project goals. Managed weekly sprints and user stories while streamlining team workflows for efficient delivery. Oversaw component integration and set up React deployment to ensure a smooth development pipeline.

B. **Anjali Singh:** Front-end Developer that led the development for the disaster analytics dashboard, implementing interactive visualizations, filtering tools, and responsive design to enhance user experience. Helped create the homepage layout, color scheme, and initial Top 10 Severe Disaster Posts table. Containerized the data pipeline scripts (data.py and genAI.py) and deployed them on a AlmaLinux VM and automated daily cron jobs. Established the data ingestion process by researching the Bluesky API, designing keyword-based tweet collection, and creating the data.py script to extract and preprocess disaster-related posts.

C. **Michelle Sang:** Front-end Developer who designed and set up the initial structure of the website using Figma and React. Contributed beyond frontend to further develop the

genAI.py script to remove the need for CSV uploads and automatically format and push data directly to the database. Analyzed and compared two geolocation APIs by evaluating the accuracy of latitude/longitude outputs and provided clear metrics for integration.

D. **Shraddha Patel:** Back-end Developer who designed and implemented the interactive DisasterMap. The DisasterMap visualizes timely disaster data with a heatmap interface. Integrated geospatial data with frontend, developed filters for users to explore DisasterMap, and configured map behavior for efficiency. Implemented custom scroll behavior and zoom handling as well as CSS tailoring.

E. **Hephzibah Sujoe:** Data scientist who experimented with and implemented a Generative AI pipeline to classify disaster-related tweets using LLMs. The system identifies disaster type, extracts a location string, and scores severity based on multiple impact metrics, with location data geocoded via OpenStreetMap.

III. Prior Products

The concept of using LLMs for sentiment analysis to determine fake news or genuinity is fairly common. There are prior versions of similar products, such as the Kaggle notebook titled "Step-by-step analysis of disaster tweets," which focused on classifying tweets as disaster-related or not. While this notebook demonstrated the fundamental techniques—data exploration, cleaning, transformation, and model building—it only analyzed static, historical data. In contrast, our project is a live web application that monitors and processes timely disaster-related tweets. Our platform goes beyond simple classification by evaluating severity levels, presenting disasters on an interactive map and other analytics tools, delivering actionable insights about ongoing global events. The Kaggle project served as a valuable proof of concept, but our solution advances this foundation into a fully operational system that provides timely, accessible intelligence through a user-friendly interface.

Summary of Expected Outcomes

In today's fast paced digital world, the ability to interpret information as it quickly comes and goes is critical but remains a major challenge due to the overwhelming volume of unstructured social media data like BlueSky. This project addresses that gap by providing a dynamic, efficient map and a feed for most recent, high-priority events. At times when traditional reporting lags behind, users and aid groups can immediately visualize where disasters are unfolding, assess their severity, and make faster decisions that are better informed about safety and response strategies. The geospatial view of incidents and direct aid requests allows emergency services to target and deploy resources quickly, and the analytics dashboard supports long-term decision making by showing trends. This platform bridges the gap between social signals and data-driven guidance, strengthening the effectiveness of first responders during critical events.

Chapter 2: System Architecture and Technical Framework

Project Framework, Theory, and Assumptions

I. Core Technical Components

A. Front-End: The front end was developed using React.js, leveraging Recharts for data visualizations and libraries like rc-slider and react-select to create an interactive, filterable dashboard. The disaster analytics dashboard includes dynamic multi-line charts, stacked bar breakdowns, severity distributions, and top location visualizations. Each visualization dynamically updates in response to user-selected filters such as disaster type, date range, severity threshold, and location, supporting exploration of crisis trends. The recent watch feed integrates directly with Supabase to retrieve posts in real time, has a dynamic filter system, and utilizes progressive loading to optimize load time and user experience.

B. Back-End: Python scripts (data.py, genAI.py) connect to the Bluesky Developer API to collect disaster-related tweets using keyword-based search logic. These tweets are preprocessed—cleaned, normalized, and filtered—and stored in Supabase, a cloud-hosted relational PostgreSQL database.

C. Modeling: Data modeling includes structured fields such as disaster type, severity score, timestamp, and inferred location, with LLM-based post-processing planned for location inference. Daily automation is handled through cron jobs on an AlmaLinux VM, where Docker containers execute the data pipeline scripts on schedule to keep the dataset continuously updated.

D. Data Source: The data pipeline was developed using Python scripts to establish a real-time connection with the BlueSky Developer API, collecting posts containing disaster-related keywords across 15+ categories. Raw tweets are preprocessed to remove non-English content, sanitize text, and standardize formatting. Extracted metadata fields include tweet ID, timestamp, disaster type, hashtags, and severity score. Cleaned data is stored in a Supabase PostgreSQL database, ready for front-end consumption. Disaster type normalization ensures consistent categorization across varied tweet language and formats. The data ingestion flow is fully automated via scheduled scripts, enabling continuous and efficient updates.

II. Alternative Designs and Approaches

A. Modeling Design Alternatives: We initially considered training our own large language model (LLM) using frameworks like Hugging Face and Llama. However, after evaluating the significant computational resources and time involved in fine tuning an LLM from scratch, we decided to take a more efficient approach by utilizing prompt engineering with existing pre-trained models. This allowed us to customize behavior and extract

relevant information from our data source without requiring the overhead of full model training.

B. **Data Source Experimentation:** Originally, we planned to use Twitter's API; however, due to access limitations and budget constraints, we opted for BlueSky's API, which offered greater accessibility and flexibility for our project needs.

C. **Analytics Dashboard Changes:** Initial designs for the Analytics Dashboard considered static charts and minimal filtering options. These were iteratively replaced with more scalable and interactive alternatives, including a normalized disaster-type taxonomy and a dynamic date-range navigation system. To better manage clutter with many disaster types, legends were redesigned to paginate after 8 items. In addition, pagination (3 days per view) was included with compact left/right navigation buttons to make the display of data less cluttered.

Chapter 3: Data Acquisition and Preprocessing

Data Input and Preparation

Our group did not train our own large language model (LLM) for the disaster response system. Instead, we adopted a prompt engineering approach with pre-existing trained models. This strategy allowed us to leverage the extensive knowledge and capabilities already embedded in established LLMs while customizing their outputs through carefully designed prompts to meet our specific disaster response needs. Prompt engineering provided several advantages as we were able to iterate and refine prompts based on performance and adapt to different disaster scenarios without retaining an LLM.

I. Pre-existing Data Sources

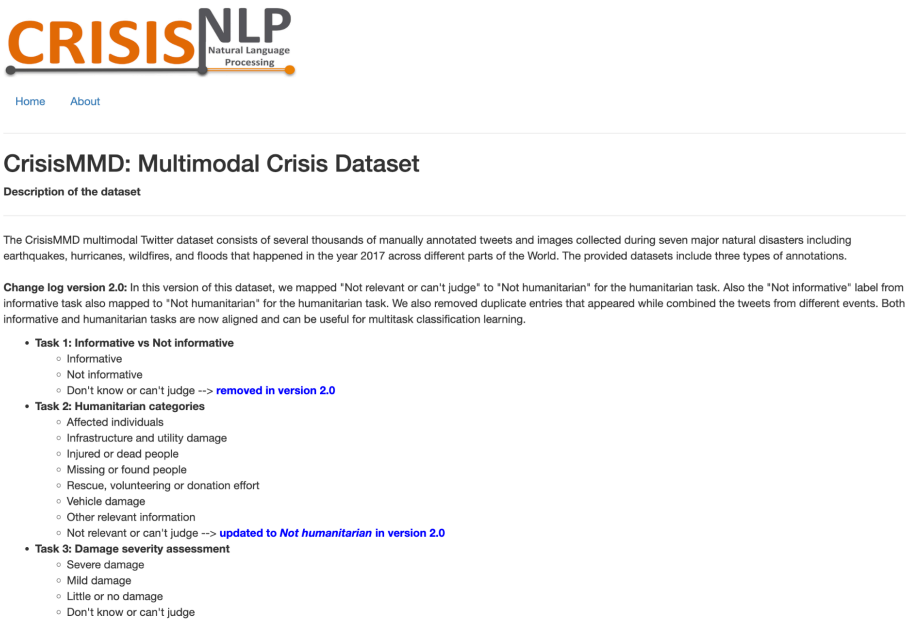
While waiting for our prompt engineering implementation to be completed, we utilized the CrisisMMD dataset as dummy data for integrating our frontend and backend components. CrisisMMD is a dataset containing natural disaster related social media posts with categories for informative vs non informative content and disaster severity levels. This dataset was an excellent provisional solution as it:

A. Integrates our frontend and backend components
B. Establishes data pipelines before finalizing prompt engineering implementations
C. Tests the system's ability to process and respond to various disaster-related content
D. Validates that our prompt engineering approach could correctly identify disaster types

Once our prompt engineering approach was complete, we transitioned from the dummy data to live data processing.

II. Disaster Types

We began with the CrisisMMD dataset, which includes labeled social media posts about disasters like earthquakes, wildfires, and floods. Using these categories as a base, we created our own comprehensive list of disasters and crisis types for the project. We added more disaster types such as heatwaves, snowstorms, landslides, and tornados to cover a broader range of events. In addition to disaster names, we also included crisis-related keywords like help, evacuation, emergency, and shelter to expand our ability to detect posts where people may be seeking or offering assistance. This helped improve the reach and relevance of our keyword-matching system when pulling posts from Bluesky.



CrisisMMD Website

Data Source and Real-Time Collection

The final project's data is collected via the BlueSky Developer API, using finalized keyword-based queries to track 15+ disaster categories. A data pipeline was implemented that ingests raw JSON responses, transforms them into CSV format, and stores structured tweet data in Supabase. Metadata fields include tweet ID, timestamp, disaster type, location (if detected), severity score, and filtered hashtags. Our application pulls real-time and historical disaster-related tweets using the Bluesky API, storing data in Supabase.

Bluesky's public API: https://api.bsky.app/xrpc/app.bsky.feed.searchposts?q={keyword}

Preprocessing Pipeline

The data.py script performs preprocessing tasks including:

- Filtering out non-English tweets (using ASCII-encoding checks)
- Emoji “demojization” and text sanitization
- Removal of special characters, URLs, and mentions
- Standardization of whitespace and casing

Processed data is passed to the multiprocessing_genai.py script for classification using an LLM. The system supports structured input for disaster relevance, type, and severity scoring, then writes clean data to Supabase.

Automation

To enable automated updates, a cron job was configured on a personal cloud server (AlmaLinux OS). The job runs two scripts daily: data.py at 10 PM and multiprocessing_genai.py at 11 PM. These handle real-time tweet collection and LLM-based labeling respectively. A Docker container was created to encapsulate the runtime environment, ensuring consistent execution on the VM.

Chapter 4: Model Development and Experiments

Model Development and Evaluation

I. Algorithms and Model Types Explored

A. Model Types - We experimented with LLMs for text classification. We benchmarked two different LLMs supporting formatted responses: Llama 3.2 and Mistral-Nemo. We did not have time to experiment with other models that support formatted responses, outside of the ones provided by OllamaClient.

B. Algorithms

1. Prompt-based Inference using Llama 3.2 - The pipeline uses LLM-based prompt engineering to guide the model to:
    a) Detect whether a tweet reports a real natural disaster
    b) Extract the disaster type and location
    c) Score the severity of the disaster based on several metrics
    d) Uses prompt engineering and few-shot learning

2. Formatted Response Algorithm - To ensure consistent, structured outputs from LLMs by enforcing a response schema defined with Pydantic BaseModel. This schema is converted into a JSON-compatible toolset and passed into the Ollama API, prompting the model to respond with structured JSON that is automatically parsed and validated.

3. Multiprocessing for Parallel Tweet Analysis - A process pool is created to map the classification function to different tweets simultaneously. Each process runs independently, enabling concurrent LLM calls.

4. Safe Retry for Fault Tolerance - If the LLM API call fails (due to timeouts or malformed JSON), the system retries up to 3 times. A retry loop is implemented with a sleep delay.

5. Geocoding Algorithm - Convert from LLM-outputted string location to location coordinates using a geocoder API. Experimented with OpenStreetMap and Geocode.xyz. Implemented techniques such as string-splitting to geocode complicated strings.

II. Evaluation Metrics

A. Parallel Processing

1. Without parallel processing, the time taken to process 100 tweets was about 2 hours and 20 minutes. With parallel processing, it takes about 1 hour.

III. Model Comparisons and Final Model Choice

Model	Disaster Type	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)	Count
llama3.2	Wildfire	96.7	95.2	100	97.5	120
	Hurricane	97.0	96.5	100	98.2	300
	Earthquake	97.6	96.8	100	98.4	250
	Flood	81.3	82.8	93.0	87.6	80
	Weighted Avg	**95.0**	**94.9**	**98.9**	**97.0**	**750**
mistral-nemo	Wildfire	94.2	92.9	98.7	95.7	120

	Hurricane	92.0	91.5	99.6	95.4	300
	Earthquake	95.2	93.9	100	96.9	250
	Flood	85.0	84.6	96.5	90.2	80
	Weighted Avg	**93.4**	**92.0**	**99.3**	**95.3**	**750**

We chose Llama 3.2, which had the highest performance metrics.

IV. Experiment Setup

A. Experiment Calculations

a. We benchmarked our two models using the following metrics: accuracy, precision, recall, and F1 score.
   i. Accuracy = (TP+TN)/(TP+FP+TN+FN)
   ii. Precision = TP/(TP+FP)
   iii. Recall = TP/(TP+FN)
   iv. F1 score = 2 x [(Precision x Recall)/(Precision + Recall)]

b. Each model was benchmarked for the different disaster types: wildfire, hurricane, earthquake and flood. Each disaster type has a different benchmark dataset. Please find the results attached: Model Benchmarking Results

B. Steps for How LLM Generates Output

a. **Initial Classification:** The tweet, including any hashtags, is passed through a prompt to determine whether it pertains to a genuine disaster event.
b. **Disaster Identification:** If the tweet is classified as a genuine disaster, it is then processed by a second prompt to extract both the disaster type and the reported location.
c. **Severity Assessment:** The tweet is analyzed through a third prompt to generate a severity score, based on evaluation metrics such as impact on daily life, infrastructure, casualties, and emergency response.
d. **Geolocation:** The extracted location string is submitted to OpenStreetMap for geocoding to obtain precise location coordinates.

C. Prompt Engineering

a. Our project demonstrates effective prompt engineering using a pre-trained LLM. We used the following prompt engineering tactics:
   i. Clear role definition
   ii. Well-defined classification rules

iii. Few-shot learning
iv. Structured final task
v. Avoidance of ambiguity

b. Our three prompts:
   i. Test for Genuineness

```
prompt0 = f"""
You are a social media analyst who is an expert on natural disaster recovery.
Determine whether the following tweet genuinely reports an ongoing or recent natural disaster.

### Classification Rules:
✅ **Classify as True** if the tweet:
- Provides specific details about an ongoing or recent natural disaster, including locations,
victims, warnings, emergency response, aid efforts, or official updates.
- Is a news headline or report about a real natural disaster, even if it doesn't contain personal narratives.
- Mentions verifiable entities (e.g., government officials, emergency agencies) discussing natural disaster impact or response.
- Is NOT discussing a natural disaster that happened in the past.

❌ **Classify as False** if the tweet:
- Uses metaphorical language (e.g., "This traffic is a tornado").
- Mentions a natural disaster in a non-literal way, such as referencing past events without new developments.
- Is purely emotional, symbolic, or does not provide any verifiable disaster-related information.

### Examples:

1. **Tweet:** "California wildfires: winds die down, helping containment efforts https://t.co/3asXQsQZgM https://t.co/b89SKa3Tuw"
   **Output:** True ✅ *(Specific disaster details, containment efforts mentioned)*

2. **Tweet:** "Flood Death rate increases in Sri Lanka has been published on Liveonchennai -
https://t.co/GmM2ENO8Mb https://t.co/WYe0eGoZlw"
   **Output:** True ✅ *(Verifiable disaster impact-death toll rising)*

3. **Tweet:** "Blue heart yellow heart please help flood social media with this message"
   **Output:** False ❌ *(No disaster details-just symbolic language)*

4. **Tweet:** "the day over and the adrenaline of a job well done flooding their bodies and minds the sisters celebrate"
   **Output:** False ❌ *(No disaster details-just symbolic language)*

### Final Classification Task:
Now, classify the following tweet:
**Tweet:** "{tweet_text}"
**Output:** (True/False)
"""
```

   ii. Identify Disaster Type and Location

```
prompt1 = (
        "You are a social media analyst who is an expert on natural disaster recovery."
        "\nA user inputted this tweet:\n"
        f"'{tweet_text}' "
        "\n\nBased on only the tweet information, answer the following questions:"
        "\n1. What type of natural disaster occurred? You must provide a brief justification first. "
        "If a disaster type is specified in the tweet text, output the disaster type, else output 'Not Specified'."
        "\n2. What is the location of the natural disaster? You must provide a brief justification first. "
        "If a location is specified in the tweet text, output a map-friendly location, else output 'Not Specified'. "
        "\nDo not make up facts. Only analyze based on the tweet text provided. "
    )
```

iii. Severity Assessment

```
prompt2 = (
        "You are a social media analyst who is an expert on natural disaster recovery. "
        "Do not make up facts. Only analyze based on the tweet text provided. "
        "\nA user inputted this tweet:\n"
        f"'{tweet_text}'"
        f"\n\nNatural Disaster Type: {disaster_type}. "
        "\nBased on only the tweet information, answer the following questions:"
        "\n1. Assess the impact of the natural disaster specified in the tweet on daily living on a scale from 0 to 10 where 0 is no impact "
        "and 10 is complete disruption of daily life. You must provide a concise justification on the impact on daily living first before providing the score. "
        "\n2. Assess the impact of the natural disaster specified in the tweet on infrastructure from 0 to 10 where 0 is no impact "
        "and 10 is life-threatening infrastructure damage. You must provide a concise justification on the impact on infrastructure first before providing the score. "
        "\n3. Assess the loss of life specified in the tweet on a scale from 0 to 10 where 0 is no loss of life and 10 is a death toll greater than 5. "
        "This means that if the death toll is greater than 5, the score must be 10. "
        "You must provide a concise justification on the loss of life first before providing the score. "
        "\n4. Assess the need for emergency response measures on a scale from 0 to 10 where 0 is no need for emergency responses "
        "and 10 involves mandatory evacuations and rescue efforts. You must provide a concise justification on the emergency responses score first before providing the score. "
    )
```

D. Sample Output Data

Tweet ID	Timestamp	Tweet	Genuine Disaster	Disaster Type	Location	Severity Score	Latitude	Longitude
4.72E+13	2025-04-03T18:34:11+00:00	march 31 2025 martano italy during the night a strong hailstorm in martano covered the streets with ice blocking the city center about 20 centimeters of hail fell	TRUE	Hailstorm	Martano, Italy	4.5	40.20253	18.30025

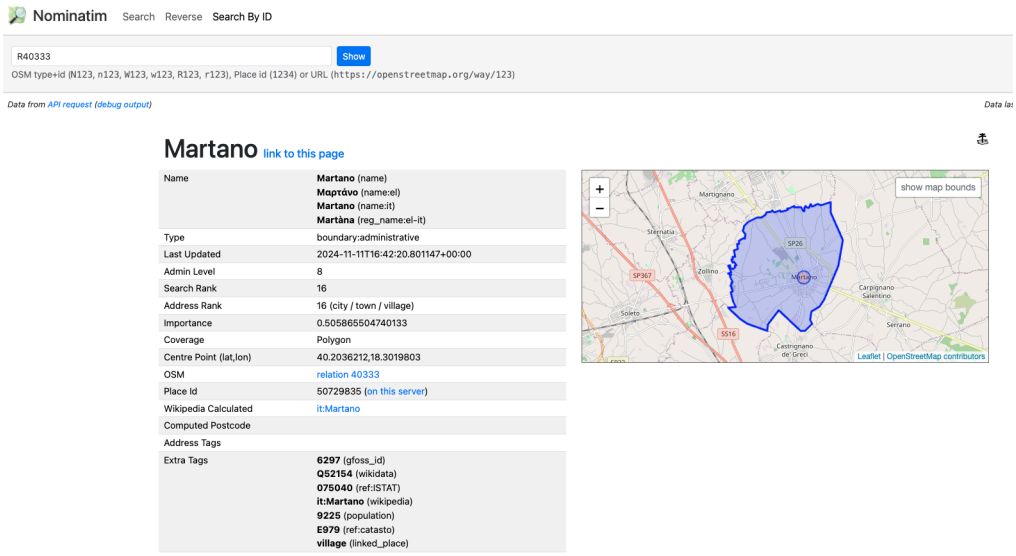
Location API and Evaluation

OpenStreetMap vs Geocode.xyz

We only had enough data to compare the two APIs on April 1st, 2025. Geocode.xyz gave us 56 records, and OpenStreetMap gave us 68. There were 35 records that matched, which we used for evaluation.

API	Accuracy (%)	Average Percent Error (%)	Count
OpenStreetMap	82.66	Latitude: 5.99 Longitude: 14.54	35
Geocode.xyz	51.43	Latitude: 18.52 Longitude: 47.41	35

OpenStreetMap was significantly better in terms of accuracy and the average percent error, which was calculated by finding the difference between the measured and true latitudes and longitudes, then divided by 360 to be normalized. Anything over 1% would be considered falsely labeled, as 1% is approximately 250 miles, which is a significant difference. Please find the results attached: Location API Benchmarking Results



Example of OpenStreetMap pulling latitude and longitude for Martano, Italy

Chapter 5: System Implementation and Deployment

We built the front end of our application using React, focusing on delivering a dynamic, interactive, and user-friendly experience. To support rich data representation, we integrated Recharts for building responsive and customizable data visualizations, and Leaflet for rendering interactive geographic maps. For UI components and styling, we used RC-Slider to create intuitive sliders and Framer Motion to enhance the application's animations and transitions. In addition to these, we leveraged a variety of other libraries to handle routing and interaction with the Supabase backend, ensuring a seamless user experience.

Front-End Visualizations

I. **Heatmap:** The Disaster Risk Heatmap provides users with a visual representation of ongoing disaster hotspots based on the data collected from BlueSky. Leaflet is our choice of the map layer.
   A. **Gradient Color System:** The Heatmap utilizes a gradient color system from blue to red to symbolize the severity of a disaster.
   Date Filtering Capabilities: Users are able to filter by dates and navigate the map to different locations in order to obtain and analyze information.

II. **Top 10 Severe Disasters of the Day:** This feature is designed to provide users with a focused view of the most urgent and impactful disasters reported. Features include:
   A. **High-Severity Filtering:** Displays the top 10 posts with the highest severity scores, determined by our large language model (LLM), based on data from the most recent date available in the database.

B. **Concise Information Display:** Each entry includes the post content, location, disaster type, severity, and time.

III. **Disaster Analytics Dashboard:** Visualizes disaster information derived from Bluesky posts. Major components include:

A. **Disaster Trends Over Time:** A multi-line chart visualizing the volume of disaster-related posts per type across selected dates, with pagination to manage large datasets.

B. **Disaster Type Breakdown:** A stacked bar chart providing a daily breakdown of disaster types, replacing earlier, less informative static charts.

C. **Top Affected Locations:** A vertical bar chart ranking locations by tweet volume and average severity, color-coded by risk level (green for low, yellow for moderate, red for high).

D. **Severity Distribution:** A stacked bar chart categorizing disasters based on low, moderate, and high severity scores.

Each chart is fully interactive, dynamically responding to disaster type selection, severity thresholds, and date ranges. Additional UX enhancements include dynamic legends that paginate after 8 items to reduce visual clutter and tooltips providing contextual insight for each chart element.

IV. **Recent Watch Feed:** This feature displays the most recent disaster-related tweets and provides timely updates for the public. Features include:

A. **Disaster Details Filtering:** The user can filter recent tweets by the disaster type, severity score, and location, narrowing down disasters to specific locations so the user can be aware of what's going on in specific communities.

B. **Color-Coded Posts:** Since each post is color-coded, users can immediately see which areas are affected the hardest from their search.

V. **First Responder Dashboard:** The first responder dashboard is designed to assist with public help requests submitted through the homepage. Major features include:

A. **Help Request Submission:** Users can submit their name, current location, phone number, type of emergency and a brief description of the situation.

B. **Emergency Verification:** Submitted information is cross-referenced with verified incident data. A confidence score is assigned based on the accuracy of the emergency and location match:

1. **Low:** Neither the emergency type nor the location matches known data
2. **Medium:** Either the emergency type or location matches
3. **High:** Both emergency type and location match existing reports

C. **Secure Access for First Responders:** Only verified first responders with approved accounts can log in to access the dashboard, protecting sensitive information from public access.

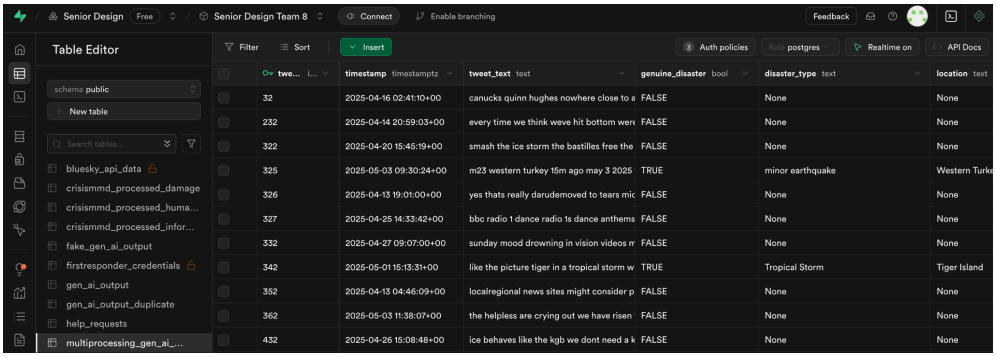
D. **Request Management:** First Responders can view all help requests along with their confidence scores and update the status of each request as pending, in progress, or resolved.
E. **Filtering and Organization:** Requests can be filtered by status, disaster type and time frame to improve situational awareness and streamline response coordination.

Backend Implementation

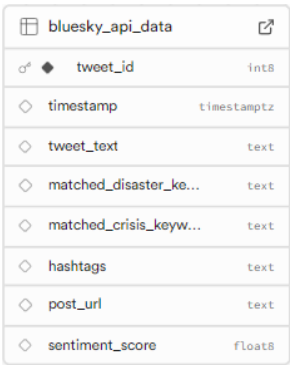
Our project does not use a traditional backend application framework. Instead, the backend is structured around a script-driven architecture, with Supabase and Python scripts handling the core processing tasks. One of the primary scripts, data.py, fetches and preprocesses data from Bluesky, then stores it in a Supabase table. Another key script, multiprocessing_genai.py, performs intensive AI analysis on this stored data—adding classifications, estimated locations, and severity scores to enrich the dataset.

Database Design

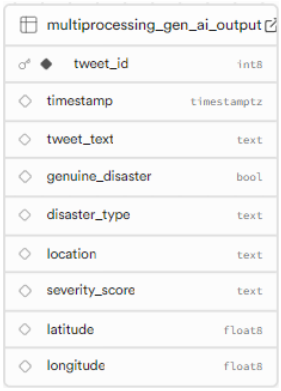
We utilized Supabase as our primary platform for database management. Supabase is an open-source backend solution that features a built-in PostgreSQL database, serverless edge functions, and automatically generated APIs for seamless integration.Our backend Python scripts connect to Supabase through its database API to efficiently store and retrieve data as needed. Meanwhile, our frontend React application interacts with Supabase's API to fetch and display the necessary information on the dashboard. We leveraged the PostgreSQL database hosted by Supabse to store all application data.



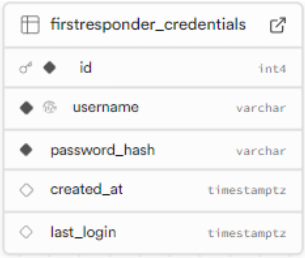
Sample Screenshot of Supabase Table Editor



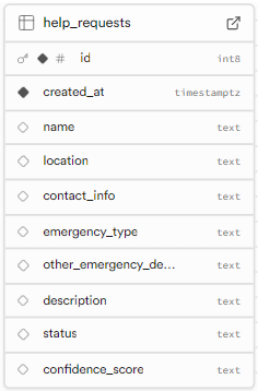
Schema for bluesky_api_data table, which stores cleaned data pulled from bluesky, acts as input for multiprocessing_genai.py script for scoring.



Schema for multiprocessing_gen_ai_output, which stores scored output from LLM, used as input for our react app.



Schema for firstresponder_credential table, which stores verified first responder accounts to access the first responder dashboard to view submitted help requests.



Schema for help_requests table, which stores submitted help requests to be displayed on first responder dashboard

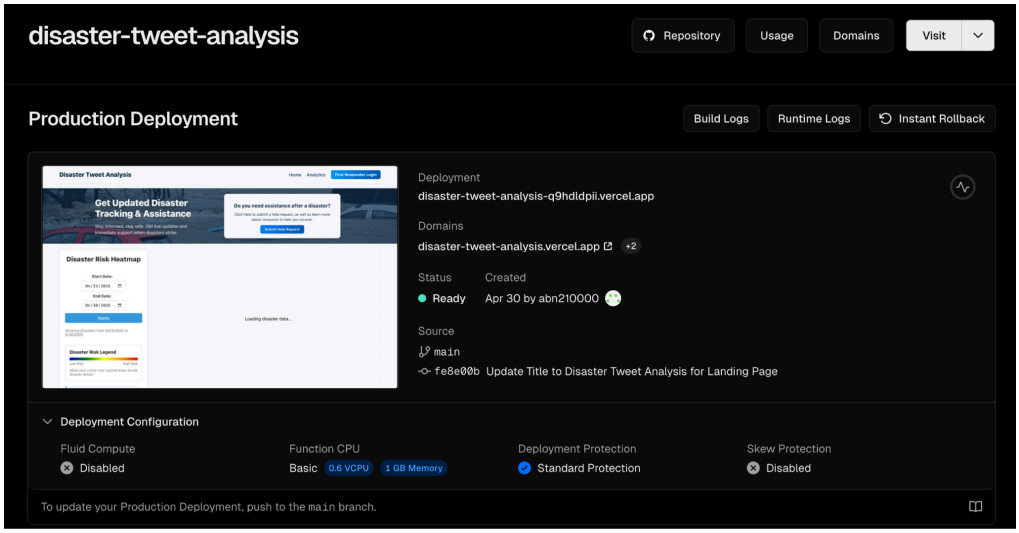
F. Server Structure: While servers are involved in the backend architecture, our project does not utilize a traditional, continuously running custom application server. Instead, the Python scripts are designed as executables, scheduled to run once daily for data processing. Supabase provides managed backend infrastructure, hosting our PostgreSQL database and exposing RESTful APIs. Additionally, our scripts interact with an Ollama server process, which hosts the Llama 3.2 model. The Ollama client within our scripts sends requests to this server to perform AI analysis during processing.

Deployment and Automation

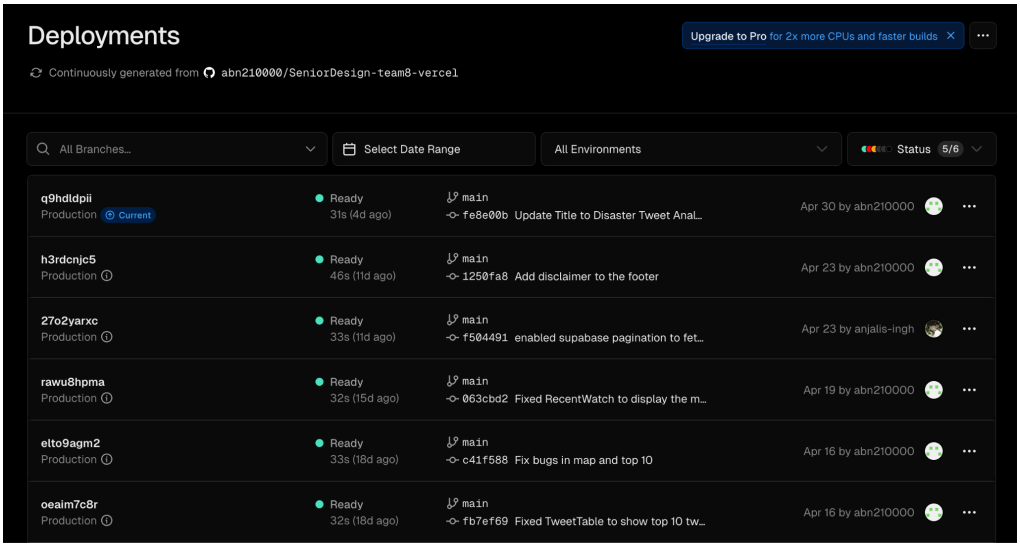
Front-end Vercel Deployment Infrastructure

The React application was deployed using Vercel, a cloud platform optimized for frontend frameworks. This deployment process was directly integrated with our GitHub repository, enabling automatic deployment upon pushing updates.

We encountered a challenge when the team member assigned to deploy the React app wasn't the GitHub repository owner. Since the project was a personal repository, they didn't have deployment access on Vercel. To resolve this, the team member forked the repository to their own GitHub account and deployed the app from there. This workaround allowed the deployment, but required the team member to manually sync changes from the original repository with every push.



Vercel Project Page



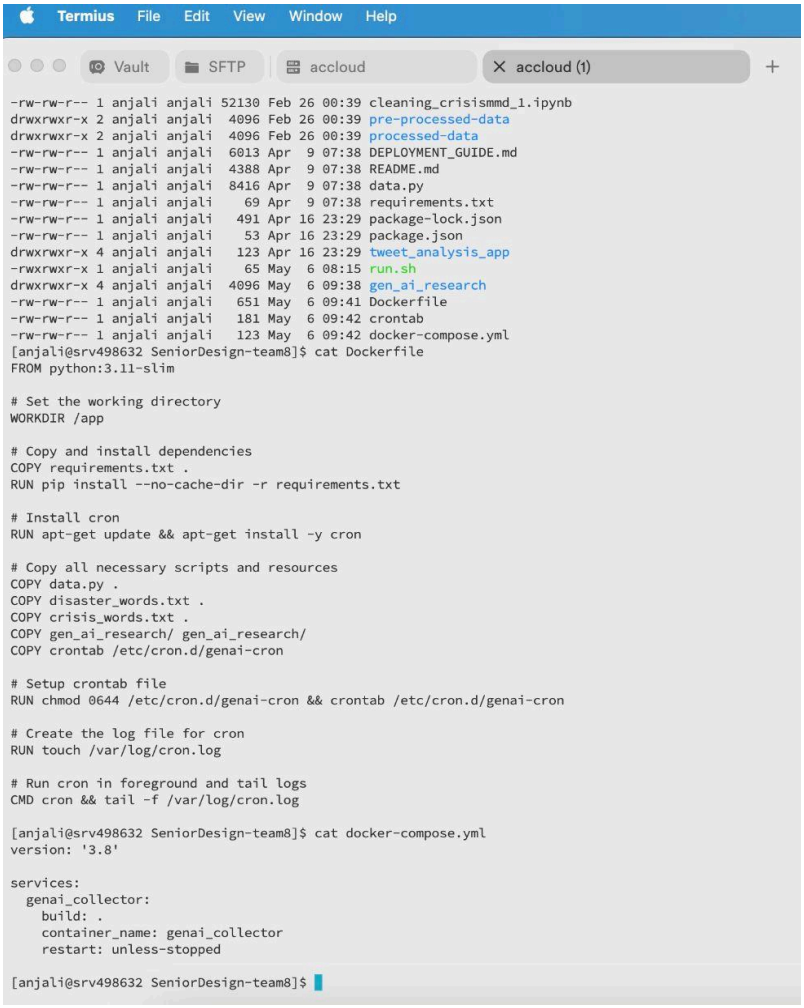
Vercel Deployment Page

Back-end Cloud Deployment Infrastructure

The backend deployment is built on a containerized architecture using Docker, hosted on a shared AlmaLinux VM. Python scripts data.py for tweet collection and preprocessing and multiprocessing_genai.py for AI classification are packaged into a Docker container to encapsulate all dependencies and ensure portability across environments.

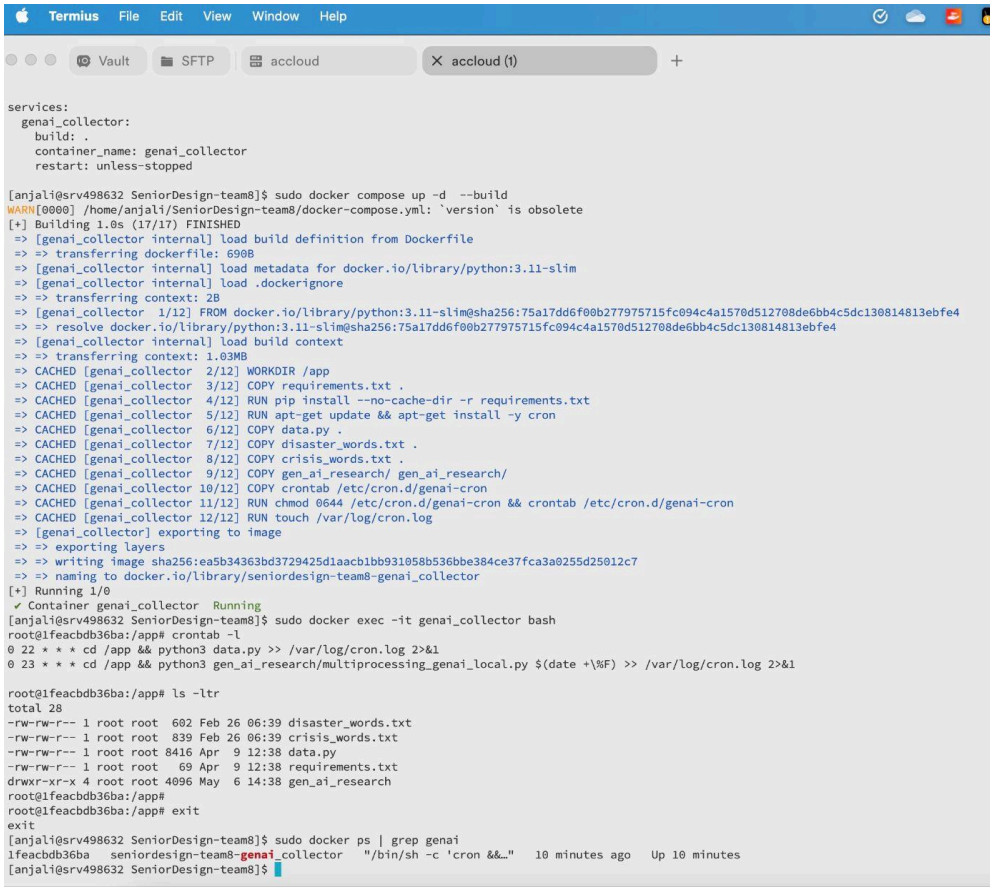
To maintain up-to-date data, two cron jobs run daily on the VM: one executes the data ingestion pipeline, and the other performs disaster classification, location inference, and severity scoring. These automated tasks fetch posts from the BlueSky API, process them inside the container, and store the enriched results in Supabase. This design reduces manual intervention while improving consistency and resilience.

The image below shows the cron job and logging setup within the container. A safe retry mechanism is used with logs written to /var/log/cron.log. This ensures that all automated LLM-based disaster analysis tasks are logged, monitored, and retried if necessary, enhancing fault tolerance and transparency in execution.



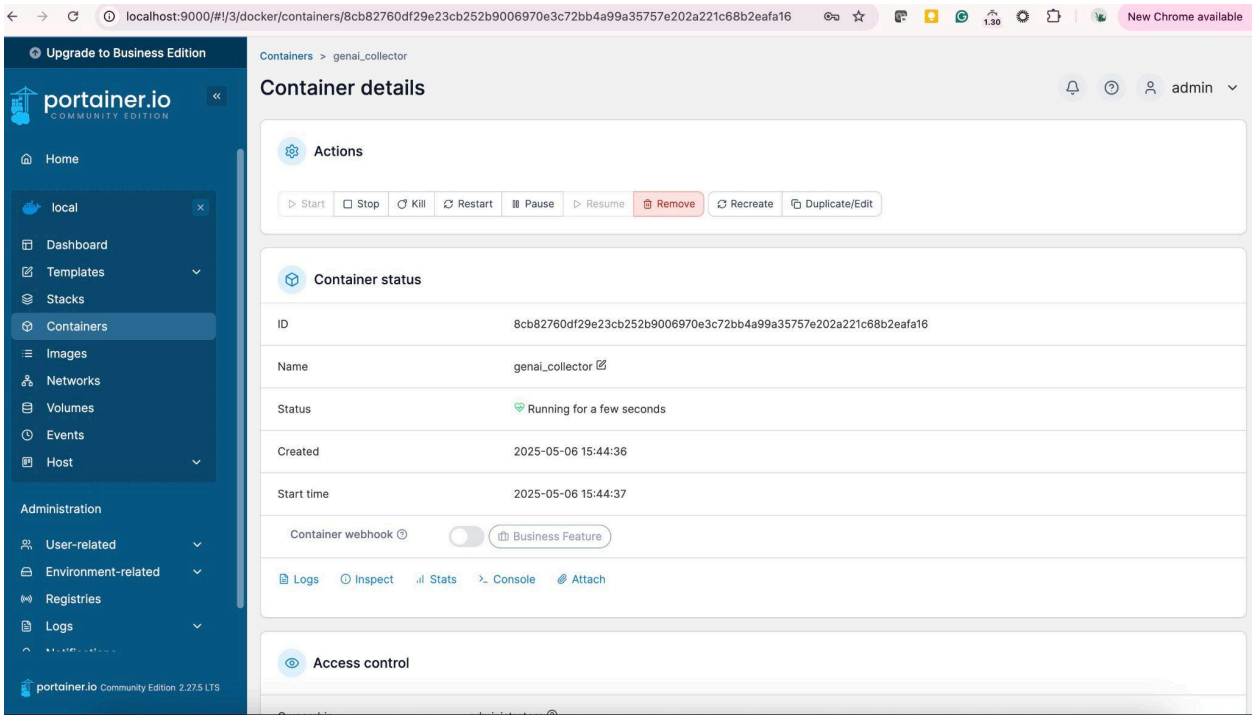
Terminal view confirming that the cron jobs and genai_collector container are actively running, along with output logging

The screenshot below highlights the infrastructure setup. The Dockerfile handles installation of dependencies, environment setup, and cron configuration. Meanwhile, docker-compose.yml orchestrates container startup, restart policies, and service naming. This setup allows streamlined deployment, monitoring, and future scaling of the backend pipeline.



Dockerfile and docker-compose.yml configuration for containering tweet ingestion and classification scripts

After deploying the container, we used Portainer Community Edition to monitor the container runtime environment. As shown below, the genai_collector container is running successfully, confirming that the backend scripts are actively deployed and executing on schedule within the Docker container.



Portainer dashboard showing the live genai_collector container running successfully

Chapter 6: End-to-End Solution Design and Product Functionality

Solution Design

I. End-to-end Design

There are three main tiers to the design: data collection and cleaning, AI processing, and the interactive front-end.

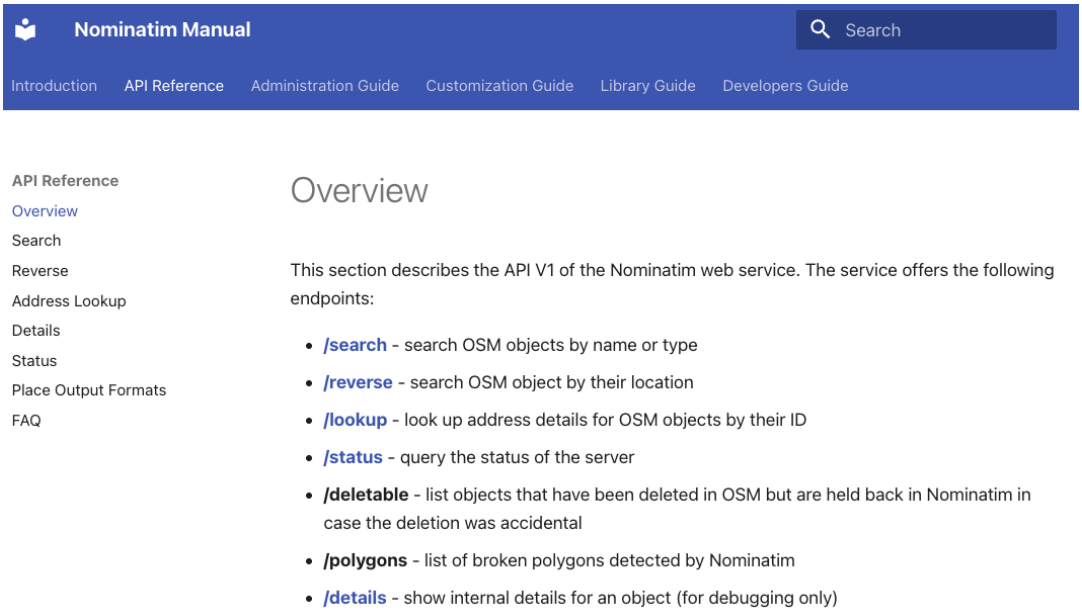
Tweets are initially collected through Bluesky API filtered by using keywords specifying more than 15 different types of disasters. Tweets are then stored in a Supabase table and undergo processing which filters out non-English words, special characters, URLS, and emojis. This process provides us with a cleaned dataset to work with further.

The AI processing takes the cleaned data and extracts key details. Using prompt engineering and LLM help us determine whether a tweet relates to a genuine disaster event, the disaster type, location, and severity score. These attributes are all determined by the LLM and are then stored into Supabase as well. Multiprocessing techniques are also in use at this phase to simultaneously

analyze multiple tweets. Location strings extracted by the LLM are translated into their latitude and longitude coordinates using the OpenStreetMap API.

```python
def get_location_data(city_name):
    url = f"https://nominatim.openstreetmap.org/search?q={city_name}&format=json&limit=1"
    response = requests.get(url, headers={'User-Agent': 'BlueskyDisasterAnalysis/1.0'})
    if response.status_code == 200 and response.json():
        return response.json()[0]
    return {}
```

Code for extracting longitude and latitude using OpenStreetMap API.



API Documentation for extracting longitude and latitude

Once data is processed and key details are extracted, the data can then be displayed on our React.js user interface. The frontend dynamically retrieves data from Supabase and visualizes it through several different displays, including a heatmap rendered through Leaflet, analytics dashboard, recent watch for tweets, and informative graphs.

These are the three tiers that bring together a pipeline from data collection to an interactive website.

II. Integration of Components

The user interface is designed prioritizing functionality and user experience while integrating our data from Supabase. When users first visit the page, they are met with a heatmap, “Disaster Risk Heat Map”, that visualizes detected ongoing crises. There are filters which allow users to narrow down their view by date. The Analytics Dashboard gives users the opportunities to further gain

insights from our dashboard. Users are able to see trends over time, investigate specific disaster types, filter by severity levels or location, and analyze KPIs.

The Top 10 Tweets Table highlights tweets of the utmost urgency from the current day. This gives users an overview of key information like locations, severities, and time of event. The Recent Watch section provides users with an interface to observe or "keep watch" of any developing stories. The Help Request Form allows individuals to quickly submit emergency details to receive assistance. On the other side of this is the First Responder Dashboard which allows emergency personnel to view and manage help requests.

Batch Job Scheduling

Batch jobs are run daily; data.py runs at 10 PM and multiprocessing_genai.py runs at 11 PM. This scheduling was chosen to balance the sheer amount of data being collected with limitations of our virtual machine environment. Because our hardware setup has some performance limitations, running the jobs too frequently could slow down the system or lead to processing issues. With a daily schedule, we are able to stay up to date with disaster events while balancing performance and data quality.

```python
# Function to fetch Bluesky posts
def fetch_bluesky_posts(keyword):
    url = f"https://api.bsky.app/xrpc/app.bsky.feed.searchposts?q={keyword}"

    try:
        response = requests.get(url)
        response.raise_for_status()
        data = response.json()

        posts = data.get("posts", [])
        results = []

        for post in posts:
            author = post.get("author", {}).get("handle", "Unknown")
            text = post.get("record", {}).get("text", "No content").lower()
            raw_timestamp = post.get("indexedAt", "Unknown")

            # Convert and filter by timestamp (last 24 hours)
            try:
                post_timestamp = datetime.strptime(raw_timestamp, "%Y-%m-%dT%H:%M:%S.%fZ")
                if post_timestamp < cutoff_time:
                    continue  # Skip posts older than 24 hours
                formatted_timestamp = post_timestamp.strftime("%Y-%m-%d %H:%M:%S")
            except ValueError:
                formatted_timestamp = raw_timestamp

            hashtags = " ".join([word for word in text.split() if word.startswith("#")])
```

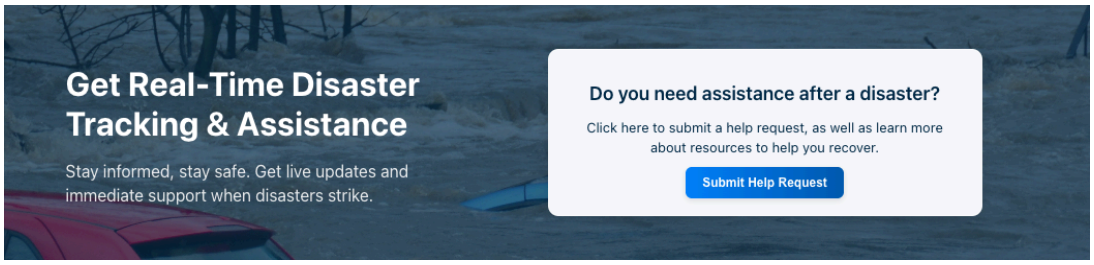
A snippet of data.py which fetches tweets from BlueSky.

```python
def fetch_from_supabase(input_date):
    supabase = get_supabase_client()

    response = (
        supabase
        .table("bluesky_api_data")
        .select("*")
        .gte("timestamp", f"{input_date}T00:00:00+00:00")
        .lt("timestamp", f"{input_date}T23:59:59+00:00")
        .execute()
    )

    data = response.data
    df = pd.DataFrame(data)
    df.to_csv("test.csv", index=False, encoding='utf-8')

    # Use proper encoding when reading the CSV file
    with open('test.csv', newline='', encoding='utf-8') as csvfile:
        reader = csv.reader(csvfile)
        row_count = sum(1 for row in reader) - 1
        print("Number of rows:", row_count)

    with open("test.csv", "r", encoding='utf-8') as csvfile:
        csv_reader = csv.reader(csvfile)
        tweet_list = list(map(tuple, csv_reader))
```

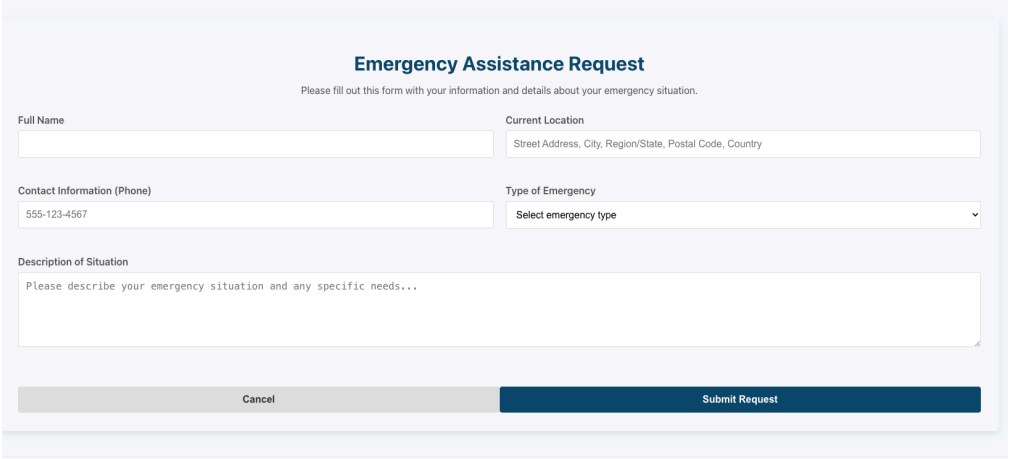
A snippet of multiprocessing_genai.py which uses the BlueSky data to further extract important details.

Functionality of Product

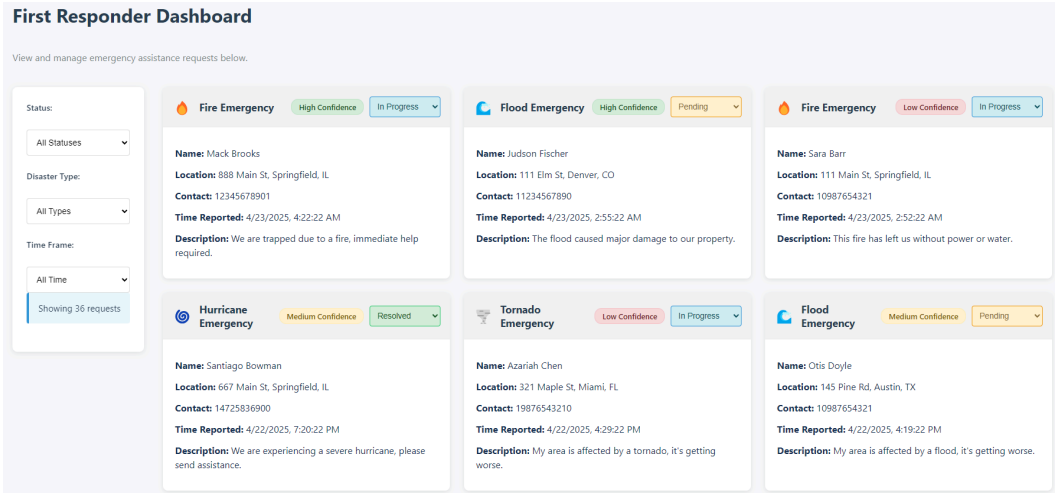
I. User interface functionalities with some screenshots and explanations



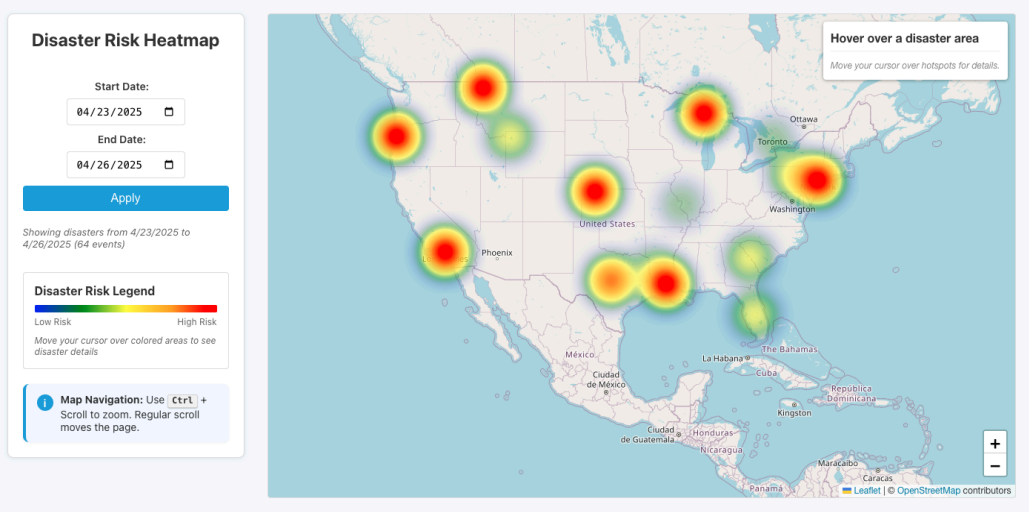
The home page provides an immediate option for users to submit a help request following a disaster, ensuring rapid access to support resources.



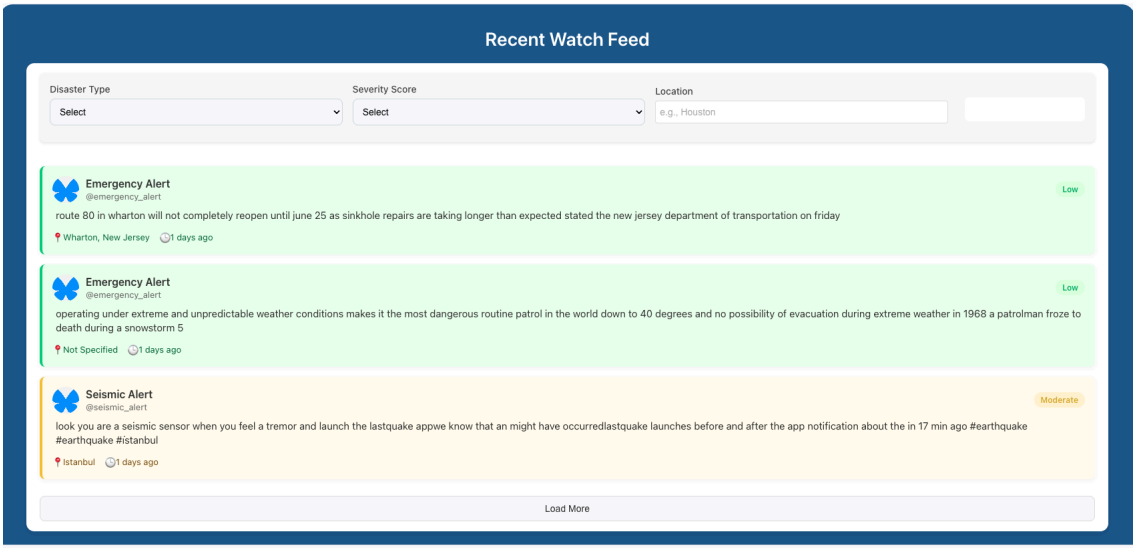
The Emergency Assistance Request form collects essential details from users seeking help, including name, location, contact information, emergency type, and a description of the situation.



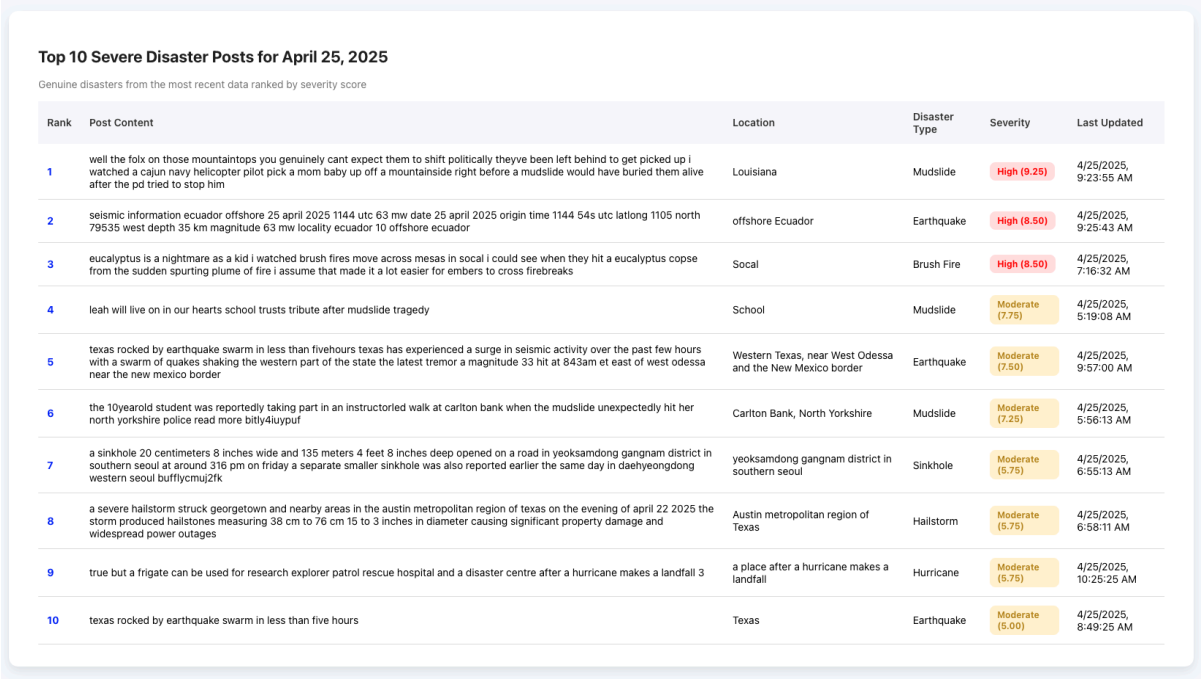
The First Responder Dashboard allows authenticated emergency personnel to view, filter, and manage incoming help requests. Each request includes details like location, contact information, reported time, and a description of the situation, along with confidence levels and request statuses.



The Disaster Risk Heat Map visualizes disaster concentrations geographically, using severity-based coloring to show areas of greatest risk. Filters allow users to adjust date ranges for analysis.



The Recent Watch Feed presents a live stream of verified disaster-related tweets, providing updated situational awareness. Users can filter the feed by disaster type, severity score, and location.

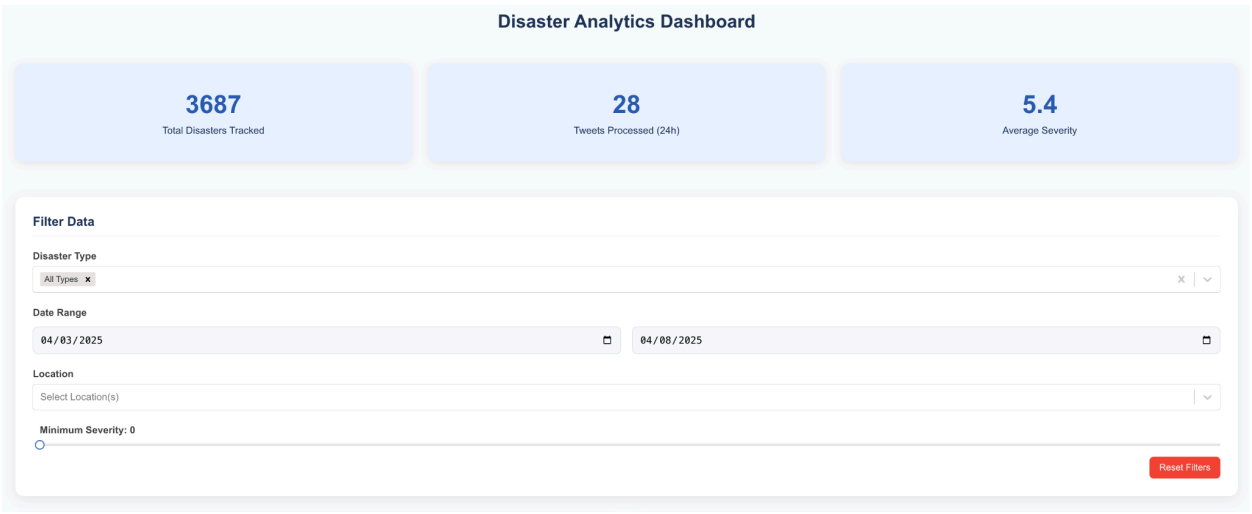


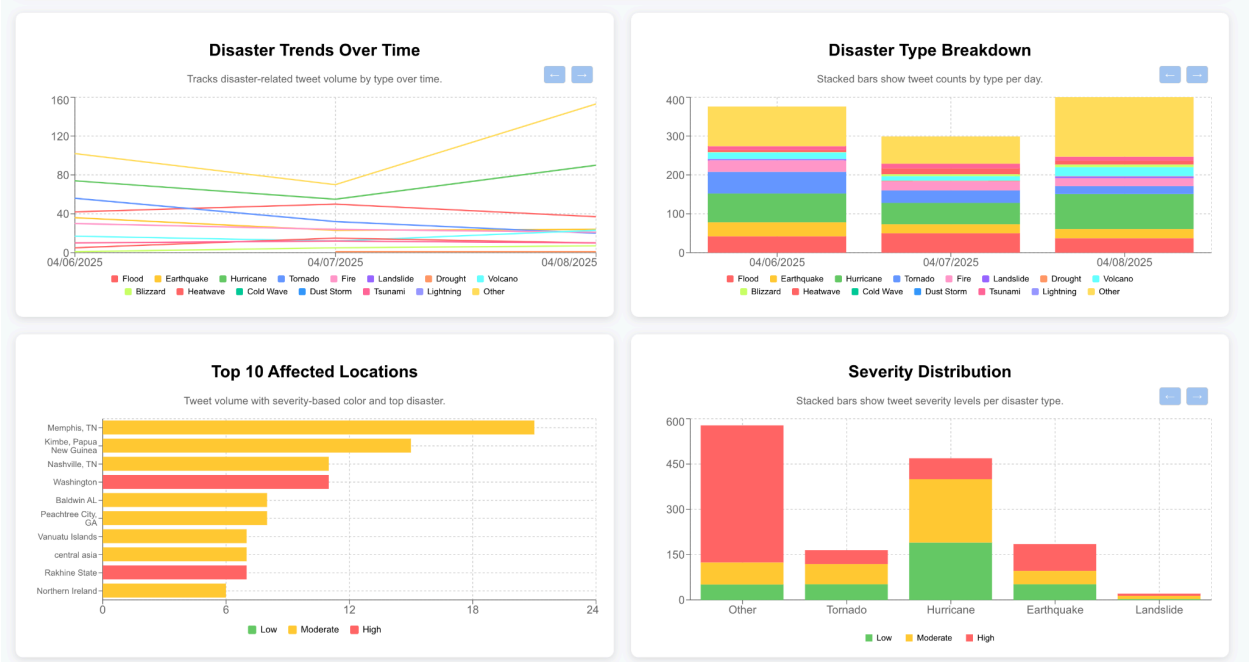
Top 10 Severe Disaster Posts for April 25, 2025

Genuine disasters from the most recent data ranked by severity score

Rank	Post Content	Location	Disaster Type	Severity	Last Updated
1	well the folx on those mountaintops you genuinely cant expect them to shift politically theyve been left behind to get picked up i watched a cajun navy helicopter pilot pick a mom baby up off a mountainside right before a mudslide would have buried them alive after the pd tried to stop him	Louisiana	Mudslide	High (9.25)	4/25/2025, 9:23:55 AM
2	seismic information ecuador offshore 25 april 2025 1144 utc 63 mw date 25 april 2025 origin time 1144 54s utc latlong 1105 north 79535 west depth 35 km magnitude 63 mw locality ecuador 10 offshore ecuador	offshore Ecuador	Earthquake	High (8.50)	4/25/2025, 9:25:43 AM
3	eucalyptus is a nightmare as a kid i watched brush fires move across mesas in socal i could see when they hit a eucalyptus copse from the sudden spurting plume of fire i assume that made it a lot easier for embers to cross firebreaks	Socal	Brush Fire	High (8.50)	4/25/2025, 7:16:32 AM
4	leah will live on in our hearts school trusts tribute after mudslide tragedy	School	Mudslide	Moderate (7.75)	4/25/2025, 5:19:08 AM
5	texas rocked by earthquake swarm in less than fivehours texas has experienced a surge in seismic activity over the past few hours with a swarm of quakes shaking the western part of the state the latest tremor a magnitude 33 hit at 843am et east of west odessa near the new mexico border	Western Texas, near West Odessa and the New Mexico border	Earthquake	Moderate (7.50)	4/25/2025, 9:57:00 AM
6	the 10yearold student was reportedly taking part in an instructorled walk at carlton bank when the mudslide unexpectedly hit her north yorkshire police read more bitly4iuypuf	Carlton Bank, North Yorkshire	Mudslide	Moderate (7.25)	4/25/2025, 5:56:13 AM
7	a sinkhole 20 centimeters 8 inches wide and 135 meters 4 feet 8 inches deep opened on a road in yeoksamdong gangnam district in southern seoul at around 316 pm on friday a separate smaller sinkhole was also reported earlier the same day in daehyeongdong western seoul bufflycmuj2fk	yeoksamdong gangnam district in southern seoul	Sinkhole	Moderate (5.75)	4/25/2025, 6:55:13 AM
8	a severe hailstorm struck georgetown and nearby areas in the austin metropolitan region of texas on the evening of april 22 2025 the storm produced hailstones measuring 38 cm to 76 cm 15 to 3 inches in diameter causing significant property damage and widespread power outages	Austin metropolitan region of Texas	Hailstorm	Moderate (5.75)	4/25/2025, 6:58:11 AM
9	true but a frigate can be used for research explorer patrol rescue hospital and a disaster centre after a hurricane makes a landfall 3	a place after a hurricane makes a landfall	Hurricane	Moderate (5.75)	4/25/2025, 10:25:25 AM
10	texas rocked by earthquake swarm in less than five hours	Texas	Earthquake	Moderate (5.00)	4/25/2025, 8:49:25 AM

The Top 10 Severe Disaster Posts table ranks the most critical disaster events based on severity, offering structured details including location, type, severity rating, and timestamps.





The Disaster Analytics Dashboard includes visualizations for disaster trends over time, disaster type breakdowns, top affected locations, and severity distributions, supporting deeper data exploration.

Future Works

There are several directions in which this project could be further developed and improved. Given more time, we would like to include more comprehensive information about the purpose of the project and its potential impact, helping users and stakeholders better understand its value and scope.

Additionally, implementing real-time data streaming, rather than relying solely on daily batch processing, is another improvement that could make the system more responsive and dynamic in fast-changing situations.

Another key area is integration with actual first responder organizations, enabling them to access and utilize the help request features to provide timely assistance to those in need. This would significantly increase the real-world impact and effectiveness of our platform.

Further, increasing the volume of processed data would enhance the demonstration of our app's capabilities, allowing for a more enhanced dashboard.

Finally, several additional features could further improve the user experience, such as implementing severity-based filtering or location-based filtering on the map to help users and responders quickly prioritize and act on critical requests.

Appendix

Final Project Link

Project Proposal

Solution Design

CrisisMMD

Supabase Documentation

Vercel Deployment Documentation

Geocode.xyz Website

OpenStreetMap Website