VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM, KARNATAKA

PROJECT REPORT

On

"Disaster Response Coordination Hub"

Submitted in partial fulfillment of the requirement for the award of the degree of

BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE AND ENGINEERING

Submitted by

USN	NAME
2SD22CS010	Anantnag N Kumbar
2SD22CS024	Deepak Kumar P S
2SD22CS036	Jeevith Jain
2SD22CS055	Poovaiah N G

Under the Guidance of

Dr. Vidyagouri Kulkarni

Dept. of CSE, SDMCET, Dharwad

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
S.D.M. COLLEGE OF ENGINEERING & TECHNOLOGY,
DHARWAD-580002

2025-2026

S.D.M COLLEGE OF ENGINEERING & TECHNOLOGY, DHARWAD –580002

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the project work and presentation entitled **"Disaster Response Coordination Hub"** *is a Bonafide work carried out by* **Anantnag N Kumbar** (2SD22CS010), **Deepak Kumar P S** (2SD22CS024), **Jeevith J Jain**(2SD22CS036), and **Poovaiah N G** (2SD22CS055)**,** *students of S. D. M. College of Engineering & Technology, Dharwad, in partial fulfillment for the award of* **Bachelor of Engineering in Computer Science and Engineering of** ***Visvesvaraya Technological University, Belgaum****, during the year 2025-2026. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report deposited in the department library. The Project has been approved, as it satisfies the academic requirements in respect of project report prescribed for the said degree.*

----------	----------
Dr. Vidyagouri Kulkarni	**Dr. U P Kulkarni**
Project Guide	HoD-CSE

External Viva

Name of Examiners	Signature with date
1. _____	
2. _____	

ABSTRACT

Flood events severely disrupt mobility and create critical information gaps, often leaving responders without real-time guidance. While existing systems provide forecasts or static maps, they rarely integrate these directly with operational tools for navigation and coordination. This project introduces the ***Disaster Response Coordination Hub (DRCH)****, a unified web platform designed to bridge this gap by linking satellite intelligence with on-ground response. Using* ***Sentinel-1 SAR*** *imagery processed via* ***Google Earth Engine****, the system automates flood detection, converting raw data into vector polygons in near real-time. These hazard layers are immediately fed into a* ***GraphHopper*** *routing engine to compute safe travel paths that actively avoid inundated areas. Accessed through a* ***React-Leaflet*** *dashboard backed by* ***Neon DB****, the platform enables authorities to manage rescue centers, verify crowdsourced reports, and assign tasks to volunteers. By consolidating detection, routing, and resource management, the system ensures that remote sensing data translates directly into actionable safety decisions.*

Contents

LIST OF FIGURES

PROBLEM STATEMENT

Despite advancements in forecasting and remote sensing, a critical operational disconnect remains between high level predictions and on the ground response. Although agencies like the Central Water Commission provide essential river forecasts and platforms like Google Crisis Map offer public awareness, a critical gap exists in operational execution. These existing systems function in isolation, delivering static advisories or high-level maps that lack the street-level precision required for safe navigation during dynamic flood events. Responders currently face a disconnected workflow, relying on fragmented data sources that cannot verify ground realities or calculate safe routes around rapidly rising waters, leading to dangerous delays and unsafe missions. The **Disaster Response Coordination Hub (DRCH)** is proposed to address this specific "last mile" disconnect by integrating disparate technologies remote sensing, hazard-aware routing, and resource management into a unified system, ensuring that intelligence directly dictates safer, faster field decisions.

CHAPTER 1
INTRODUCTION

Background and The Operational Gap

Floods have long been recognized as one of the most frequent and debilitating natural hazards facing the Indian subcontinent. The annual monsoon, while vital for agriculture, frequently brings torrential rains that overwhelm river basins and urban drainage infrastructures. The result is a recurring cycle of disruption that severs transportation networks, damages critical utilities, and isolates communities when they are most vulnerable. With the accelerating impact of climate change, these events are becoming less predictable and more intense, challenging the capacity of traditional disaster management frameworks.

However, the core challenge during these crises is often less about the water itself and more about the "information void" that follows. While national agencies like the Central Water Commission (CWC) and the National Disaster Management Authority (NDMA) provide robust high-level forecasting and preparedness guidelines, a significant disconnect remains at the operational level. When it floods, the specific, granular intelligence required for immediate action is frequently missing. Forecasting models may predict that a river will rise by two meters, but they rarely translate that data into street-level guidance. They do not tell a rescue driver which specific bridge is submerged, or which highway is safe for a supply convoy. Consequently, authorities are often forced to rely on delayed manual reports or intuition, leading to inefficient resource allocation. First responders may be deployed to areas that are already inaccessible, while the public remains unaware of safe evacuation routes, relying on panic-driven hearsay rather than verified data. This "last-mile" gap between scientific prediction and on-ground reality is where lives are often lost or endangered.

The Technological Landscape

From an engineering perspective, the tools required to bridge this gap already exist, yet they function in rigid silos. The current technological landscape for disaster response is fragmented into three distinct domains: remote sensing, routing navigation, and administrative coordination.

In the domain of remote sensing, extensive academic research has demonstrated the efficacy of Synthetic Aperture Radar (SAR), specifically from satellites like Sentinel-1, in detecting floodwaters through heavy cloud cover. Platforms like Google Earth Engine (GEE) have democratized access to this data, allowing for rapid processing of satellite imagery. However, in most existing workflows, this powerful technology ends at the "map making" stage. It produces static images or "raster masks" that are useful for post-event analysis but are unintelligible to navigation systems.

Simultaneously, routing technologies like Google Maps or OpenStreetMap are ubiquitous but generally assume a "blue-sky" scenario where the road network is intact. They do not dynamically update to reflect that a road is currently under four feet of water unless a user manually reports a traffic jam. Finally, coordination platforms for volunteers and authorities often lack geospatial integration, serving merely as contact lists or message boards rather than tactical command centers. The failure of these systems to talk to one another creates a scenario where we have eyes in the sky (satellites) and maps on the ground (routing engines), but no mechanism to connect them into a unified operational picture.

The Proposed Solution: Disaster Response Coordination Hub (DRCH)

To address these systemic inefficiencies, this project introduces the Disaster Response Coordination Hub (DRCH). This platform is not simply a visualization dashboard; it is a fully integrated operational framework designed to link satellite-derived intelligence directly with actionable field response. The system moves beyond static mapping to create a dynamic, living model of the disaster zone.

The technical core of the DRCH automates the pipeline from detection to decision. It leverages **Google Earth Engine** to ingest **Sentinel-1 SAR** imagery, applying advanced speckle filtering and backscatter differencing to detect inundation automatically. Crucially, the system converts these raster detections into vector polygons digital boundaries that define the flood zones.

These polygons are then fed immediately into a modified **GraphHopper** routing engine. This integration is the system's defining innovation: it treats the satellite-detected flood zones as "restricted areas" within the road network. When a user requests a route, the system calculates a path that actively avoids these dynamic hazards, ensuring safety in a way that standard GPS tools cannot. The entire platform is delivered via a modern web stack, utilizing **React** and **Leaflet** for a responsive user interface and **Neon DB** for a scalable, serverless backend that manages authentication and real-time data flow.

Operational Framework and Strategic Significance

The DRCH is built to unify the efforts of the three critical stakeholders in disaster management: authorities, volunteers, and the public.

For **Authorities**, the hub serves as a command-and-control centre. It aggregates data into a single pane of glass, allowing officials to view the entire theatre of operations, deploy rescue centres strategically, and verify crowdsourced reports to filter out noise. It transforms their role from reactive information gathering to proactive resource orchestration.

For **Volunteers**, the platform provides structure and safety. Instead of self-deploying to potentially dangerous areas, registered volunteers receive specific, actionable tasks assigned by authorities. Most

importantly, the hazard-aware routing protects them during their missions, reducing the risk of responders becoming victims themselves.

For the **Public**, the system offers transparency and agency. Citizens can view the extent of the flooding in their neighbourhoods and identify safe routes to the nearest relief centres, reducing panic and reliance on unverified information.

Ultimately, the significance of the Disaster Response Coordination Hub lies in its ability to reduce decision latency. By automating the flow of information from satellite sensors to street-level routing, the project aims to replace intuition with evidence, ensuring that every movement on the ground is guided by the most accurate intelligence available from the sky.

CHAPTER 2
SUSTAINABLE DEVELOPMENT GOALS

The United Nations' 17 Sustainable Development Goals (SDGs) provide a comprehensive framework that increasingly shapes how engineering projects are conceived, designed, and executed worldwide. Engineers play a pivotal role in achieving these goals, as infrastructure, technology, and innovation are fundamental enablers of sustainable development. From SDG 6 (Clean Water and Sanitation) driving water treatment facility designs to SDG 7 (Affordable and Clean Energy) influencing renewable energy projects, these goals are no longer peripheral considerations but central to project planning and evaluation. Modern engineering projects are expected to demonstrate clear alignment with relevant SDGs, often requiring environmental impact assessments, social benefit analyses, and long-term sustainability metrics that extend beyond traditional cost-benefit calculations.

Engineering projects now commonly integrate multiple SDGs simultaneously, recognizing the interconnected nature of sustainable development. For instance, a transportation infrastructure project might address SDG 9 (Industry, Innovation and Infrastructure) while also contributing to SDG 11 (Sustainable Cities and Communities) through reduced emissions and improved urban mobility, and SDG 13 (Climate Action) through low-carbon construction methods. This multi-dimensional approach requires engineers to adopt systems thinking, considering not just technical performance but also social equity, environmental protection, and economic viability throughout a project's lifecycle.

The SDG framework has transformed engineering practice by establishing measurable targets and indicators that help quantify project impacts beyond traditional engineering metrics. Projects are increasingly evaluated on their contribution to goals like SDG 3 (Good Health and Well-being), SDG 12 (Responsible Consumption and Production), and SDG 15 (Life on Land), requiring engineers to collaborate with diverse stakeholders including environmental scientists, social planners, and community representatives. This paradigm shift has elevated the profession's responsibility, positioning engineers as key agents of sustainable development who must balance technical excellence with ethical considerations, resource efficiency, and long-term societal benefit in every project they undertake.

In the context of the **Disaster Response Coordination Hub (DRCH)**, the project is specifically mapped to **SDG 11 (Sustainable Cities and Communities)**, **SDG 13 (Climate Action)**, **SDG 15 (Life on Land),** and **SDG 3 (Good Health and Well-being)**. By integrating satellite-based flood detection with hazard-aware routing, the system directly addresses the mandate of **SDG 11** to significantly reduce the number of deaths and people affected by water-related disasters, fostering resilience in urban and rural settlements. Simultaneously, it contributes to **SDG 13** by strengthening the adaptive capacity of authorities to manage climate-induced extreme weather events through real-time intelligence. Furthermore, the platform supports **SDG 3** by minimizing response times for medical emergencies and ensuring the safety of first responders during relief operations. The project also touches upon **SDG 15** by monitoring terrestrial ecosystems affected by inundation, thereby creating a cohesive engineering solution that serves both societal safety and environmental resilience.

CHAPTER 3

OBJECTIVES

The **Disaster Response Coordination Hub** project is guided by the following objectives to:

1. Develop an automated flood detection pipeline using Sentinel-1 SAR imagery and Google Earth Engine for real-time inundation mapping.
2. Implement a hazard-aware routing engine that computes safe paths by dynamically avoiding satellite-detected floods, crowdsourced reports, and manually drawn danger zones.
3. Establish a scalable data infrastructure using Neon DB to manage real-time geospatial data, volunteer registries, and incident verification workflows.
4. Create a centralized dashboard for authorities to coordinate rescue efforts, verify public reports, and assign actionable tasks to volunteers.

CHAPTER 4
LITERATURE REVIEW

4.1 Government Forecasting and Preparedness Systems

4.1.1 Central Water Commission (CWC) Forecasting Network:

The Central Water Commission (CWC) operates India's primary flood forecasting network. As detailed in their operational guidelines, the CWC monitors river water levels through a vast network of gauge stations and utilizes hydrological models to issue basin-scale forecasts and advisories to state and local authorities [1]. These forecasts are critical for early warning, allowing administration to mobilize resources days in advance of a potential breach. The system relies heavily on rainfall data and upstream flow measurements to predict water levels at specific gauge points.

- **Current Capability:** The CWC system excels at "Macro-Level Prediction". It can accurately predict that the Yamuna or Godavari River will cross the danger mark in 48 hours. This provides the necessary lead time for mass evacuations and strategic resource positioning.
- **Operational Gap:** While excellent for preparedness, the system lacks "Micro-Level Spatial Resolution". A gauge reading tells authorities that the water level is high, but it does not generate a precise map of which specific streets, bridges, or villages are currently inundated. First responders cannot use a river-level graph to navigate a relief truck. There is no mechanism to translate a CWC advisory into a navigable routing layer.
- **Our Contribution:** The DRCH complements this by focusing on the "During-Event" phase. Instead of predicting river levels, our system maps the actual extent of the water on the ground using satellite radar, providing the spatial granularity that gauge-based forecasting lacks.

4.1.2 National Disaster Management Authority (NDMA) Guidelines:

The National Disaster Management Authority (NDMA) serves as the apex body for disaster management policy in India. Their guidelines on flood management provide a comprehensive framework for pre-disaster planning, structural measures (like embankments), and non-structural measures (like zonation mapping) [2]. These documents serve as the blueprint for state and district disaster management plans, emphasizing the need for coordinated response and resilient infrastructure.

- **Current Capability:** These systems excel at "Information Aggregation" and "Public Reach". They effectively warn millions of users about general hazards and help people locate nearby

shelters. Their reliance on crowdsourced data (like traffic slowdowns in Google Maps) provides a layer of real-time responsiveness.

- **Operational Gap:** Despite their utility, these platforms are primarily "Dissemination Tools", not "Operational Tools".
  - ***Lack of Authority Control:*** Authorities cannot easily use Google Maps to assign specific tasks to registered volunteers or verify the credibility of every report.
  - ***Routing Limitations:*** Standard navigation apps are designed to optimize for traffic speed, not flood safety. They do not ingest raw satellite flood polygons to hard block submerged roads. A road might appear "clear" on Google Maps simply because no one is driving on it to report a slowdown, when it is washed away.
- **Our Contribution:** The DRCH differs by focusing on "Operational Command" and "Hazard-Aware Routing." Unlike public maps that suggest routes based on traffic, our GraphHopper engine actively prioritizes safety, refusing to route through detected flood zones regardless of traffic conditions. It also adds the layer of authority verification that open platforms lack.

4.2 Public Situational Awareness Platforms

4.2.1 Google Crisis Response / Crisis Map

In the commercial and humanitarian sectors, platforms like Google Crisis Map represent the state-of-the-art in public information dissemination. During major emergencies, these platforms aggregate diverse data sources including shelter locations, weather alerts, and crowdsourced traffic disruptions and present them on a user-friendly map interface accessible to the public [3].

- **Current Capability:** These systems excel at information aggregation and reach. They effectively warn millions of users about general hazards and help people locate nearby shelters [3]. Their reliance on user-reported data (like traffic slowdowns) provides a layer of responsiveness.
- **Operational Gap:** Despite their utility, these platforms are primarily designed for public awareness rather than authority-led operations.
  - *Lack of Authority Control:* Authorities cannot easily use public maps to assign specific tasks to registered volunteers or verify the credibility of every report.
  - *Routing Limitations:* Standard navigation apps are designed to optimize traffic speed [3]. They typically rely on traffic flow data to infer blockages rather than using raw satellite data to actively designate "hazard zones". This means a route might appear clear simply because there is no traffic data to suggest otherwise.

- **Our Contribution:** The DRCH differs by focusing on "Operational Command." It allows to designate hazard zones manually or via satellite input, ensuring that the routing engine actively prioritizes safety over speed. It also adds a layer of verification that open platforms typically lack.

4.3 Remote Sensing Methodologies

4.3.1 Sentinel-1 SAR Flood Mapping

The scientific community has extensively validated the use of Synthetic Aperture Radar (SAR) for flood detection. Martinis provides a comprehensive review of techniques using Sentinel-1, highlighting its ability to penetrate cloud cover a crucial advantage over optical satellites during monsoon floods [4]. The research demonstrates that SAR backscatter values drop significantly on smooth water surfaces, allowing for reliable water detection [4].

- **Current Capability:** The literature proves that all-weather detection is feasible. Algorithms can distinguish water from land with consistent accuracy using automated processing chains [4].
- **Operational Gap:** This research is largely academic and analytical. The typical output is a static map or a scientific paper published for analysis. The focus is often on the accuracy of the classification algorithm rather than the speed of data delivery. There is rarely a mechanism discussed to push this derived data instantly into a logistics or routing workflow.
- **Our Contribution:** The DRCH operationalizes these scientific findings. We implement proven backscatter thresholding techniques within Google Earth Engine but extend the pipeline to automatically vectorise the results. This moves the data from an analyst's screen to an operational dashboard.

4.3.2 Change Detection and Thresholding

Building on basic detection, Shen et al. propose advanced methods using Change Detection and Thresholding. By comparing a "pre-event" image with a "during event" image, this method effectively isolates floodwater from permanent water bodies, reducing false positives [5].

- **Current Capability:** Change detection provides context-aware mapping. It ensures that a normal river is not flagged as a flood, identifying only the excess water that threatens communities [5].
- **Operational Gap:** While the detection methods are robust, the integration with decision support is typically missing. A flood mask tells you where the water is, but it doesn't automatically tell a driver to avoid that area. The literature stops at the boundary of the image processing domain.

- **Our Contribution:** The DRCH utilizes this change detection methodology to ensure accurate mapping but adds the critical post-processing step. We treat the change detection output as a dynamic constraint for pathfinding algorithms, bridging the gap between remote sensing science and transportation logic.

CHAPTER 5

DETAILED DESIGN

The design phase serves as the blueprint for the Disaster Response Coordination Hub (DRCH), translating the requirements and objectives into a structured technical framework. This chapter details the system architecture, data flow patterns, database schema, and behavioral models of the application.

5.1 Architectural Components

Figure-1 illustrates the high-level system architecture of the Disaster Response Coordination Hub (DRCH), showing the interaction between the presentation layer, application logic layer, and external computational services. It highlights client–middleware communication, asynchronous flood detection via the Google Earth Engine API, and hazard-aware routing through the GraphHopper service, providing an overview of data flow and system responsibilities.

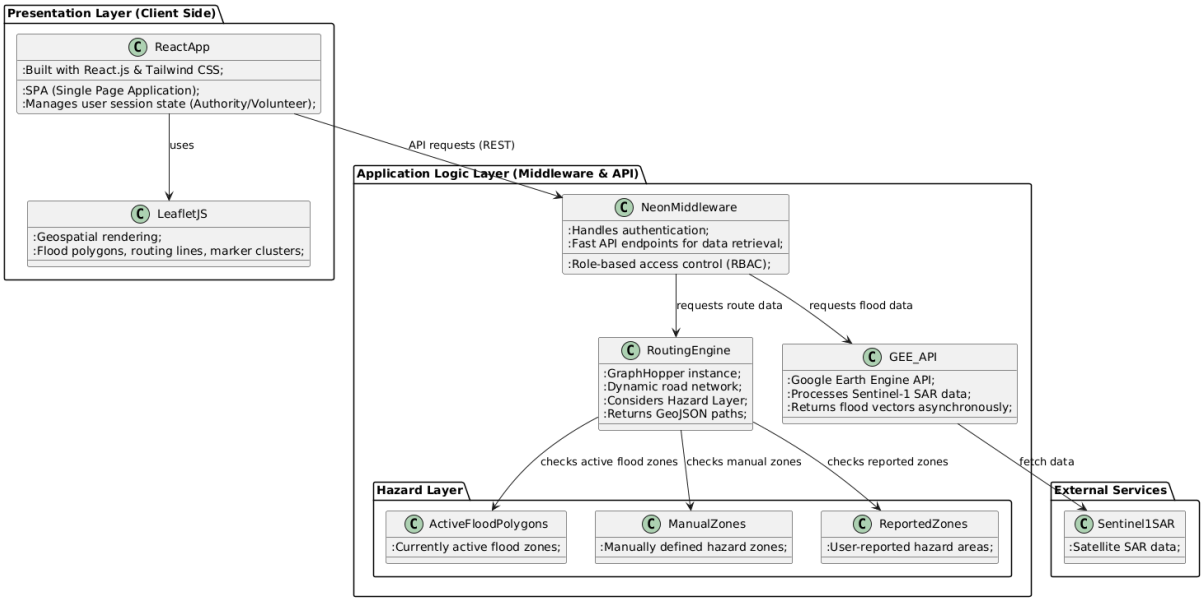

Figure-1: Architecture Diagram

5.1.1 Presentation Layer (Client Side):

- Built using **React.js** and **Tailwind CSS**, providing a responsive Application.
- **Leaflet.js** is embedded for geospatial rendering, managing the complex overlay of flood polygons, routing lines, and marker clusters.
- The client handles state management for user sessions (Authority/Volunteer)

5.1.2 Application Logic Layer (Middleware & API):

- **Google Earth Engine (GEE) API:** This serves as the computational core for flood detection. It operates asynchronously, processing Sentinel-1 SAR data on Google's cloud infrastructure and returning processed flood vectors to the main system.
- **Routing Engine Service:** A custom instance of **GraphHopper**, hosted to allow dynamic modification of the road network graph. It intercepts routing requests, checks against the current "Hazard Layer" (active flood polygons and manual zones), and returns a GeoJSON path.
- **Neon Middleware:** Handles authentication, role-based access control (RBAC), and FastAPI endpoints for data retrieval.

5.2 Data Flow Design (DFD)

Figure-2 presents the Level 0 Data Flow Diagram of the Disaster Response Coordination Hub (DRCH), illustrating the system as a central process and its interaction with external entities, including satellite data sources, users, and the map provider.

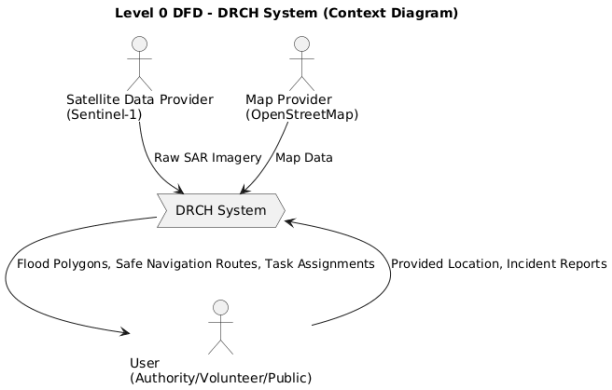

Figure-2: Level 0 Dataflow Diagram

5.2.1 Level 0 DFD (Context Diagram)

At the highest level, the system interacts with three external entities: the **Satellite Data Provider** (Sentinel-1), the **User** (Authority/Volunteer/Public), and the **Map Provider** (OpenStreetMap).

- **Input:** Raw SAR Imagery, User Location, Incident Reports.
- **Process:** The DRCH System (Central Process).
- **Output:** Flood Polygons, Safe Navigation Routes, Task Assignments.

5.2.2 Level 1 DFD (Process Decomposition)

This level decomposes the central system into three major functional modules with clearly defined data dependencies and outputs.

1. Flood Detection Module (Code-Driven SAR Pipeline)

Figure-3 illustrates the Level 1 Data Flow Diagram for the Flood Detection Module, detailing the code-driven Sentinel-1 SAR processing pipeline implemented using Google Earth Engine and the transformation of raw imagery into validated flood polygons.

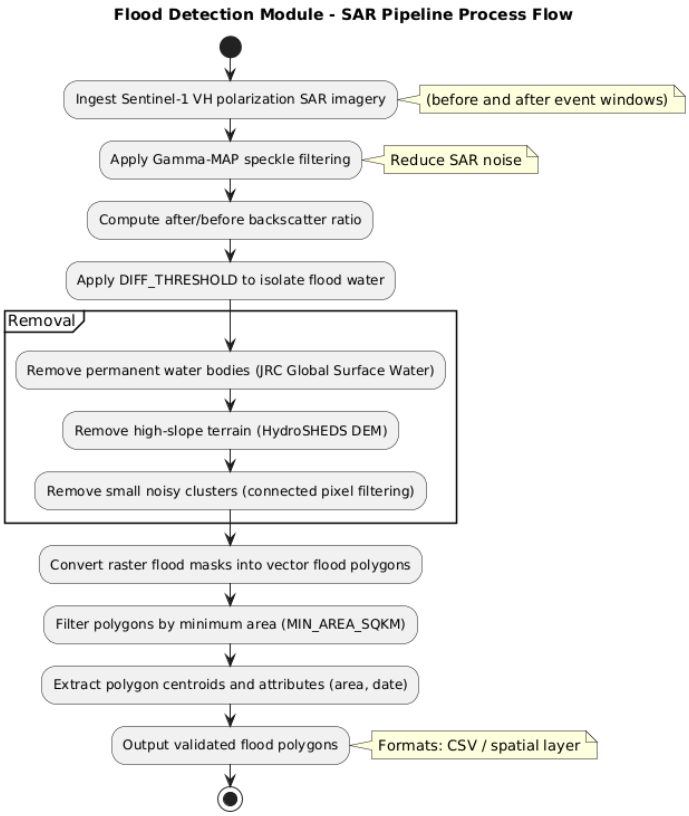

Figure-3: Flood Detection Model

This module is **directly implemented using the Sentinel-1 SAR processing pipeline defined in the provided Google Earth Engine (GEE) code**. The workflow is fully automated and parameter controlled.

Process Flow:

- Ingests **Sentinel-1 VH polarization SAR imagery** (before and after event windows).
- Applies **Gamma-MAP speckle filtering** to reduce SAR noise. Computes a**fter/before backscatter ratio** and applies a **strict threshold (DIFF_THRESHOLD)** to isolate flood water.
- **Removes:**
  - Permanent water bodies (JRC Global Surface Water)
  - High-slope terrain (HydroSHEDS DEM)
  - Small noisy clusters using **connected pixel filtering**.
- Converts raster flood masks into **vector flood polygons**.
- Filters polygons by **minimum area (MIN_AREA_SQKM)**.

- Extracts **polygon centroids and attributes** (area, date).
- Outputs **validated flood polygons** as vector data (CSV / spatial layer).

Output: Sentinel-derived flood polygons usable by routing and visualization modules.

2. Routing Module (Multi-Source Flood-Aware Path Planning)

Figure-4 depicts the Level 1 Data Flow Diagram of the Routing Module, illustrating how multiple flood polygon sources are integrated with the road network to compute flood-aware and safe navigation routes.

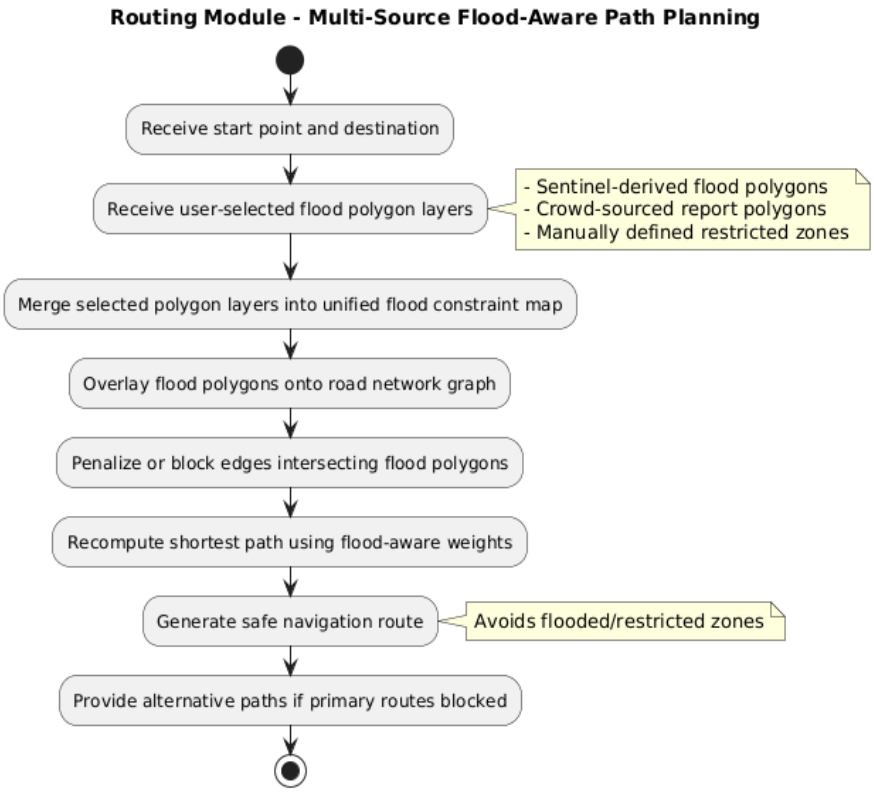

Figure-4: Routing Module

This module computes safe routes by incorporating **multiple flood polygon sources**, not just satellite detection.

Inputs: Start point and destination, User-selected flood polygon layers, **Sentinel-derived flood polygons** (from Flood Detection Module), **Crowd-sourced report polygons**, **manually defined restricted zones** (admin-created).

Process Flow:

- Merges selected polygon layers into a unified flood constraint map.
- Overlays flood polygons onto the road network graph.
- **Penalizes or blocks edges** intersecting flood polygons.
- Recomputes shortest path using flood-aware weights.

Output:

- **Safe navigation route** avoiding flooded or restricted zones.
- Alternative paths if primary routes are blocked

3. Coordination Module (Live Crowd-Sourced Reporting & Control)

Figure-5 Illustrates the Level 1 Data Flow Diagram of the Coordination Module, showing the handling of live crowd-sourced hazard reports, their immediate visualization, and subsequent moderation by authorized officials.

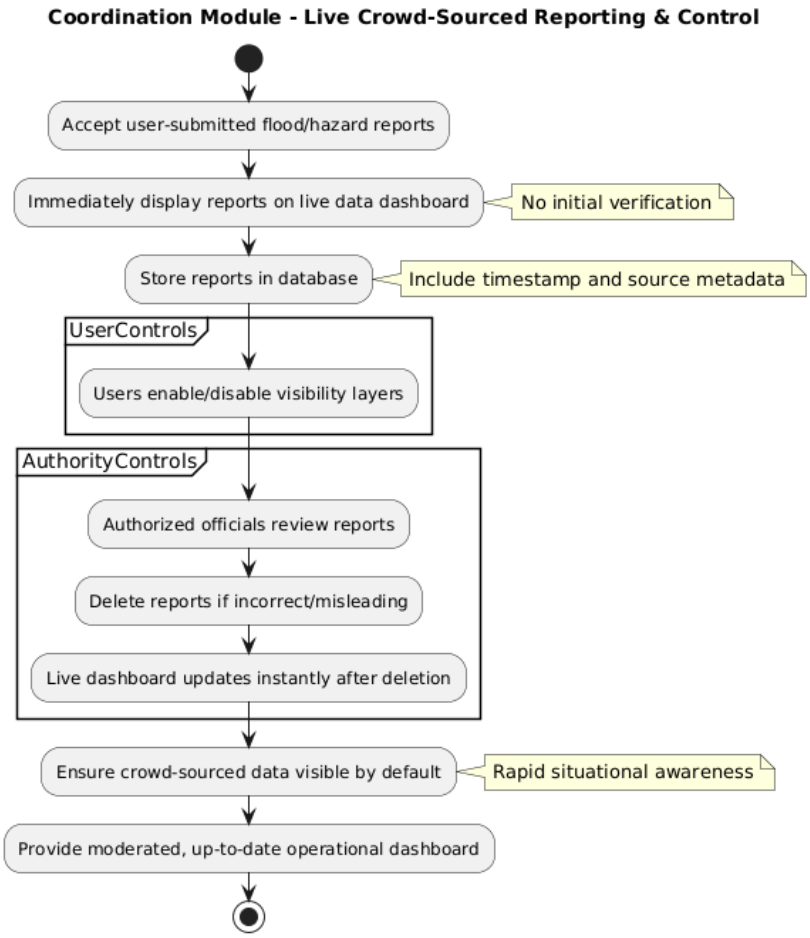

Figure-5: Coordination Module

This module manages **real-time user reports** with immediate public visibility and delayed authority moderation.

Process Flow:

- Accepts user-submitted flood or hazard reports.
- Immediately displays reports on the live data dashboard (no initial verification).
- Stores reports in the database with timestamp and source metadata.
- Provides toggle controls:
  - Users can enable/disable visibility layers.

- Authorized officials can:
    - Delete reports at any time if deemed incorrect or misleading.
- Removal instantly updates the live dashboard.

Key Principle:

- Crowd-sourced data is visible by default to ensure rapid situational awareness.
- Authority verification is corrective, not gatekeeping.

Output:

- Live, user-controlled hazard layers.
- Moderated, up-to-date operational dashboard.

5.3 Database Schema:

Figure-6 presents the database schema of the Disaster Response Coordination Hub (DRCH), illustrating the core entities and relationships supporting user management, incident reporting, task coordination, and rescue center operations.

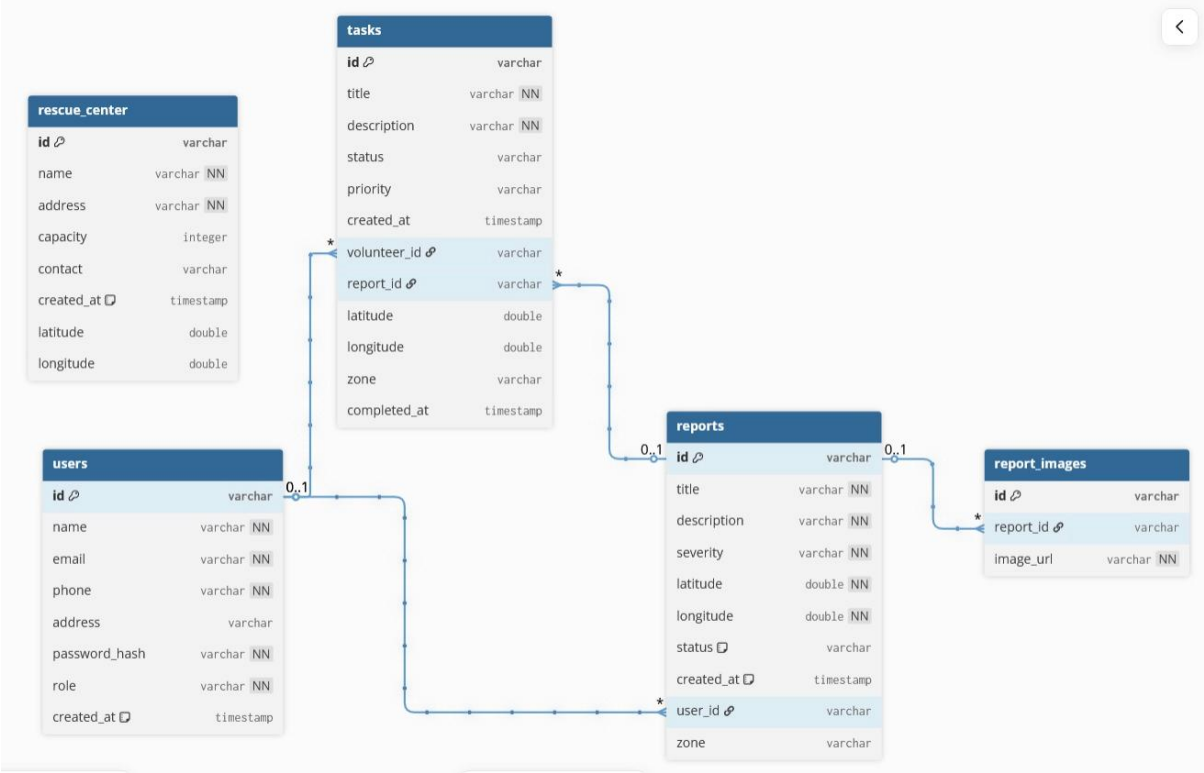

Figure-6: Database Schema

5.4 Use Case Design

The system is designed around specific actors and their goals.

5.4.1 Actors

Authority(Admin):

Oversees the system and moderates' data. Responsible for monitoring disaster status, managing flood layers, moderating crowd-sourced hazard reports (e.g., deleting the incorrect entries), and assigning resources.

Volunteer:

Field responder who receives assigned tasks, navigates safely using the system's routes, and updates their task status in real time.

Public User:

General users who consume information and contribute hazard data. They can view live hazard maps, find safe evacuation routes, report new hazards, and draw manual hazard zones.

5.4.2 Use Case

Figure-7 Illustrates the use case diagram of the Disaster Response Coordination Hub (DRCH), depicting the primary system actors and their interactions with core system functionalities.

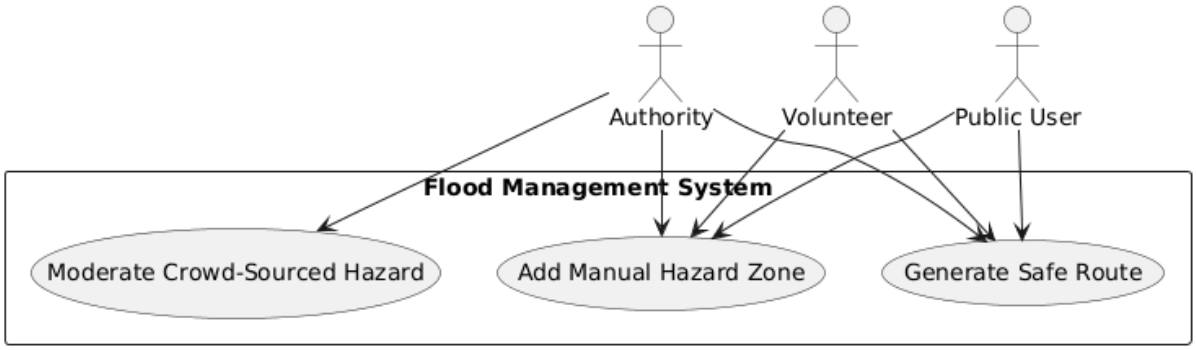

Figure-7: Use Case Diagram

5.5 Sequence

5.5.1 Safe Route Calculation

Figure-8 Shows the sequence diagram for the Safe Route Calculation process, outlining the interactions between the user, frontend, Google Earth Engine, and routing engine to generate flood-aware navigation paths.

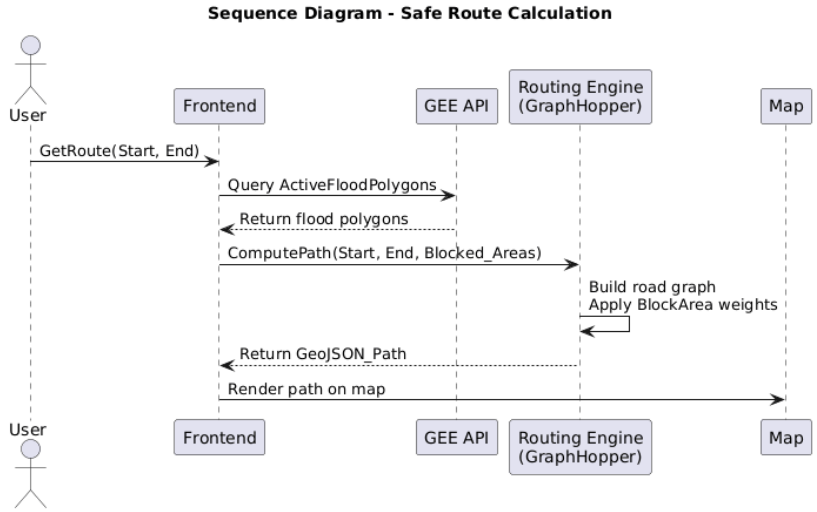

Figure-8: Sequence Diagram – Safe Route Calculation

This process is the core innovation of the project.

1. **User** sends GetRoute(Start, End) request to **Frontend**.
2. **Frontend** queries sentinel data through **GEE** for ActiveFloodPolygons.
3. **Frontend** sends ComputePath(Start, End, Blocked_Areas) to **Routing Engine (GraphHopper)**.
4. **Routing Engine** builds graph applies BlockArea weights to edges intersecting polygons.
5. **Routing Engine** returns GeoJSON_Path.
6. **Frontend** renders the path on the map.

5.5.2 Sequence: Task Assignment

Figure-9 Illustrates the sequence diagram for Task Assignment, detailing the interactions between the authority, system, database, and volunteer during the task allocation process.

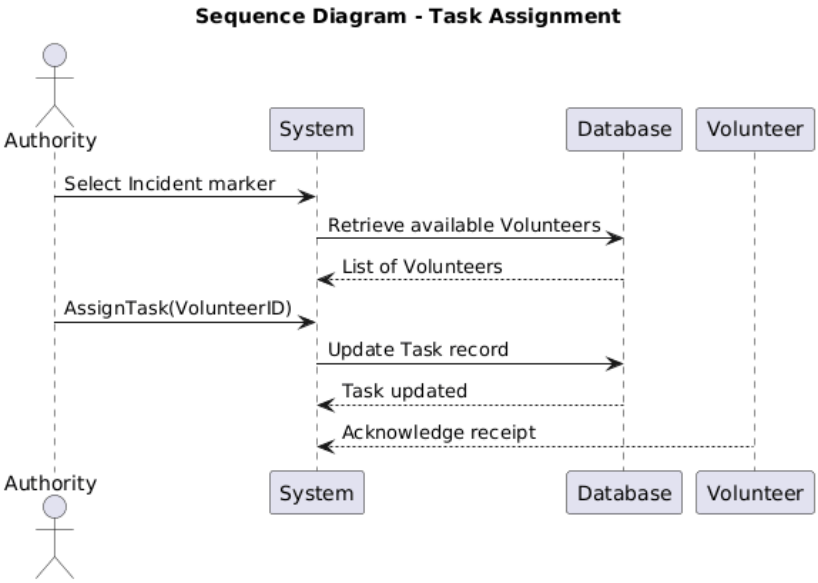

Figure-9: Sequence Diagram - Task Assignment

1. **Authority** selects an Incident marker.
2. **System** retrieves Volunteers.
3. **Authority** clicks AssignTask(VolunteerID).
4. **Database** updates Task record.
5. **Volunteer** acknowledges receipt.

5.6 Activity Design

Activity diagrams model the workflow logic, particularly where decisions are involved.

5.6.1 Activity: Flood Detection Pipeline

Figure-10 depicts the activity diagram for the Flood Detection Pipeline, modeling the workflow from manual initiation through SAR data processing to the generation of flood polygons displayed on the live map.

Start Node: Manual execution request.

Action: Authenticate and initialize Google Earth Engine.

Action: Define analysis date, thresholds, and before/after time windows.

Action: Fetch Sentinel-1 SAR collection (VH polarization, IW mode, descending orbit, 10 m resolution).

Decision: Is valid Sentinel-1 data available for both before and after periods?

- Yes → Continue
- No → End process

Action: Create mosaicked Before and After SAR images and clip to area of interest.

Action: Apply **Gamma-MAP speckle filter** to both images.

Action: Calculate backscatter **ratio (After ÷ Before)**.

Action: Apply flood threshold (ratio > DIFF_THRESHOLD).

Action: Mask permanent water bodies (JRC Global Surface Water).

Action: Mask slope > 5 degrees (exclude steep terrain).

Action: Remove small noisy clusters using connected pixel filtering.

Action: Vectorize flood raster to polygons.

Action: Filter polygons by minimum area (MIN_AREA_SQKM) and compute centroids.

End Node: Populate flood polygons on the Leaflet live map layer.

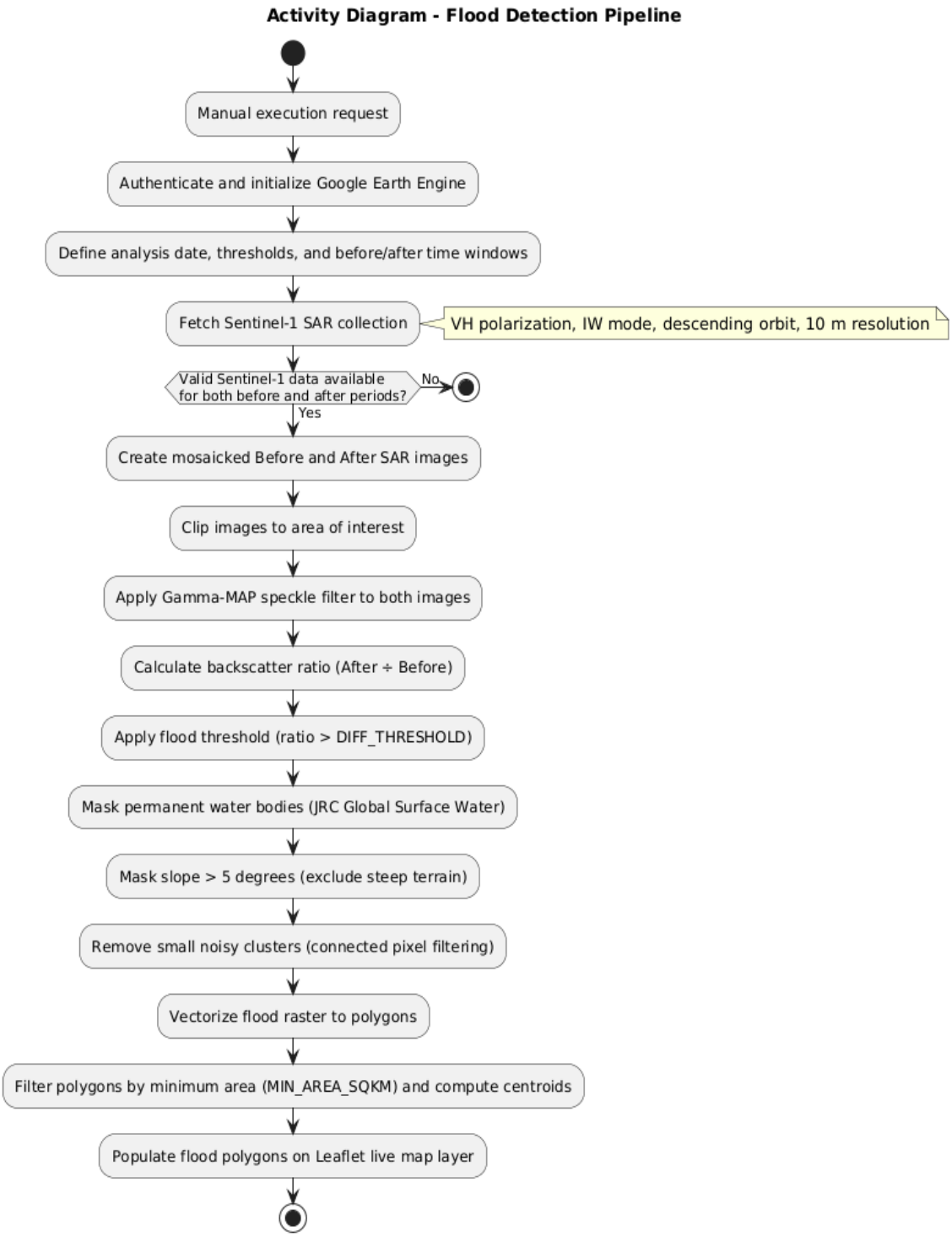

Figure-10: Activity Diagram - Flood Detection Pipeline

5.7 Component and Interface Design

Figure-11 Illustrates the routing interface and backend API interactions of the Disaster Response Coordination Hub (DRCH), showing the map dashboard layout, user controls, and key API endpoints that support flood-aware navigation and incident reporting.

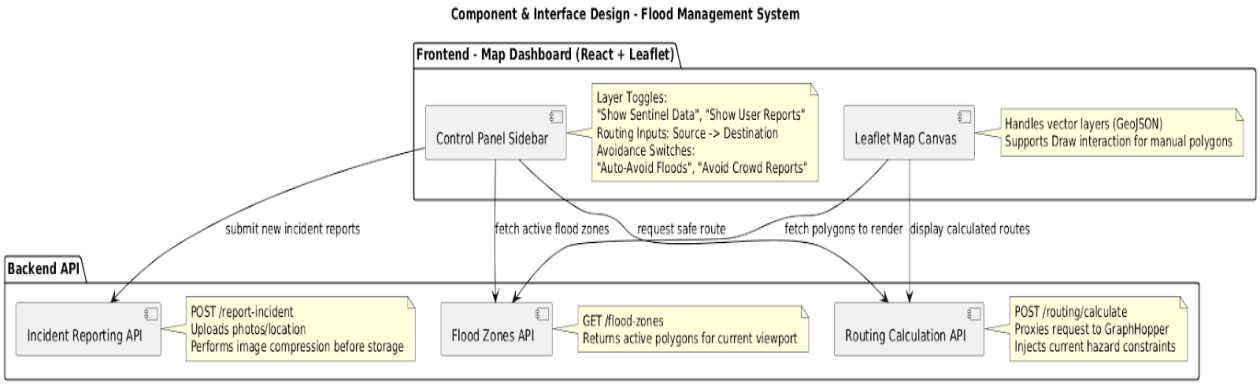

Figure-11: Components and Interface Design

5.7.1 Routing Interface (Map Dashboard)

The dashboard is split into two primary zones:

- **The Canvas:** Full-screen Leaflet map handling vector layers (GeoJSON). It supports "Draw" interaction for manual polygon creation.
- **The Control Panel:** A floating sidebar (using Tailwind CSS classes) containing:
  - *Layer Toggles:* "Show Sentinel Data", "Show User Reports".
  - *Routing Inputs:* "Sorce" to "Destination".
  - *Avoidance Switches:* "Auto-Avoid Floods", "Avoid Crowd Reports".

5.7.2 Backend API Specifications (Functions)

- GET /flood-zones: Returns active polygons for the current viewport.
- POST /report-incident: Endpoint for public users to upload photos/location. Requires image compression before storage.
- POST /routing/calculate: Proxy endpoint to the GraphHopper instance, injecting current hazard constraints into the request body.

CHAPTER 6
PROJECT SPECIFIC REQUIREMENTS

6.1. Functional Requirements

These requirements define the specific behaviours and functions the system must support to meet user needs.

6.1.1 User Management and Authentication

Role-Based Access Control (RBAC): The system must distinguish between three distinct user roles:

- **Authority:** Full access to verify reports and assign tasks.
- **Volunteer:** Access to task lists and routing; ability to update task status.
- **Public:** Read-only access to maps and routing; write access for incident reporting only.

6.1.2 Flood Detection and Mapping

Satellite Ingestion: The system must interface with the Google Earth Engine (GEE) API to automatically ingest Sentinel-1 Synthetic Aperture Radar (SAR) imagery.

Automated Vectorization: The system must convert raster flood masks into GeoJSON vector polygons without manual intervention.

6.1.3 Hazard-Aware Routing

Dynamic Graph Management: The routing engine must accept a dynamic "blacklist" of polygons (flood zones) and exclude all road edges intersecting these polygons from the pathfinding graph.

Multi-Source Avoidance: The system must allow users to toggle avoidance parameters:

- *Avoid Satellite Floods:* (On/Off)
- *Avoid Crowdsourced Hazards:* (On/Off)
- *Avoid Manual Zones:* (On/Off)

Path Visualization: The system must return the calculated route as an encoded polyline or GeoJSON LineString to be rendered on the client map.

6.1.4 Incident Reporting and Verification

Crowdsourced Submission: Public users must be able to upload an image and a short description of a local hazard. The system must automatically capture the user's geolocation.

Crowdsourced Report assignment: The dashboard will provide queue for authorities to assign and review these submissions.

6.2. Non-Functional Requirements

These requirements define the quality attributes, performance, and constraints of the system. NFR1:

6.2.1 Performance and Latency

- Routing Calculation: The system must calculate a safe route of up to 50km length within < 500 milliseconds, even when avoiding complex polygon geometries.
- Map Rendering: The frontend must render the flood polygon layer (potentially containing thousands of vertices) without freezing the browser interface.
- Database Querying: Neon DB (PostgreSQL) queries for fetching local incidents must execute in < 100 milliseconds.

6.2.2 Scalability

Graph Processing: The routing engine must be capable of loading larger OpenStreetMap datasets (e.g., expanding from District to State level) without code refactoring, provided sufficient RAM is allocated.

6.2.3 Reliability and Availability

Data Integrity: The system must ensure that no task assignment is lost during network drop; state synchronization must occur once connectivity is restored.

6.2.4 Usability

Responsive Design: The interface must be fully functional on mobile devices (for volunteers/public) and desktop screens (for command center authorities).

Intuitive Symbology: The interface utilizes a **distinctive color-coded system** to differentiate between flood zones, crowd-sourced data, manual entries, and safe routes, ensuring immediate situational awareness during high-stress operations.

6.3 Technical and Hardware Requirements

This section details the specific infrastructure required to host and run the application, with a focus on the computational demands of the routing engine.

6.3.1 Server-Side Requirements (Routing Engine)

The **GraphHopper** routing instance is the most resource-intensive component of the backend. Unlike standard web servers, it loads the entire road network graph into Random Access Memory (RAM) to ensure millisecond-response times.

- **Processor:** Quad-core CPU (2.5 GHz or higher) is recommended for fast graph traversal and contraction hierarchy preparation.
- **Memory (RAM) - Critical Requirement:**
- *Small Region (District Level):* Minimum **4 GB RAM**.
- *Medium Region (State Level e.g., Karnataka):* Minimum **16 GB to 32 GB RAM**.
- *Reasoning:* GraphHopper creates a massive in-memory graph structure from the .osm.pbf raw data. As the coverage area increases, the node/edge count grows exponentially. Insufficient RAM will lead to OutOfMemoryError or severe performance degradation due to disk swapping.
- **Storage:**
- High-speed **SSD Storage** is mandatory.
- Space required for raw OpenStreetMap data files (southern-india.osm.pbf) and the generated cache/graph artifacts.
- **Software Environment:**
- Java Runtime Environment (JRE) 11 or higher (required for GraphHopper).

6.3.2 Client-Side Requirements

Device: Smartphone (Android/iOS) or Desktop Computer.

Browser: Modern web browser with **WebGL** support (Chrome, Firefox, Safari, Edge) to handle heavy map tiling and vector rendering.

Location Services: GPS/GNSS module must be enabled on the device for "My Location" only for reporting feature.

CHAPTER 7

IMPLEMENTATION

The implementation phase transforms the architectural design into a functional software system. This chapter details the technical execution of the **Disaster Response Coordination Hub (DRCH)**, focusing on the four critical pillars: Satellite Data Processing, Geospatial Database Management, Hazard-Aware Routing, and the Client-Side Interface. The system is built using a decoupled architecture, where heavy computational tasks (satellite analysis, routing) are handled by specialized services, while the user experience is delivered through a lightweight, responsive web application.

7.1 Flood Detection Pipeline (Google Earth Engine)

The core intelligence of the system relies on the automated detection of floodwaters using Sentinel-1 Synthetic Aperture Radar (SAR) imagery. This is implemented using the **Google Earth Engine (GEE) Python API**.

7.1.1 Image Collection and Filtering

The first step involves defining the region of interest (Karnataka) and filtering the Sentinel-1 collection. We utilize the COPERNICUS/S1_GRD collection, selecting Interferometric Wide (IW) swath mode for high-resolution ground detection.

Logic: The system filters by date (Pre-Event vs. Post-Event), polarization (VH), and orbit properties.

Implementation Detail:

```
# Load Sentinel-1 Collection
collection = ee.ImageCollection('COPERNICUS/S1_GRD') \
    .filter(ee.Filter.eq('instrumentMode', 'IW')) \
    .filter(ee.Filter.listContains('transmitterReceiverPolarisation',
'VH')) \
    .filter(ee.Filter.eq('orbitProperties_pass', 'DESCENDING')) \
    .filter(ee.Filter.eq('resolution_meters', 10)) \
    .filter(ee.Filter.bounds(geometry)) \
    .select('VH')
```

7.1.2 Advanced Speckle Filtering (Gamma Map)

SAR data inherently contains "speckle" noise—granular interference that can look like random bright/dark pixels. To clean the image without blurring edge details, we implemented a **Gamma Map Filter**. This advanced algorithm utilizes local statistics (mean and variance) within a 3x3 kernel to smooth homogeneous areas while preserving the sharp boundaries of water bodies.

Algorithm Logic:

1. Compute local mean and variance.
2. Calculate the variation coefficient () and weighting function ().
3. Apply the filter expression to determine the new pixel value.

7.1.3 Change Detection and Thresholding

The flood extraction relies on the principle that water surfaces reflect radar waves away from the sensor. We implement a **Difference Approach** with strict thresholds to avoid false positives (like muddy fields).

1. **Ratio Calculation:** Compute the ratio of the filtered "After" image to the "Before" image.
2. **Thresholding:** We utilize a **Difference Threshold of 1.35**. Pixels exceeding this ratio are flagged as potential water.

```
difference = after_filtered.divide(before_filtered)
flooded = difference.gt(1.35).rename('water').selfMask()
```

7.1.4 Masking and Vectorization

To ensure accuracy, we apply three critical masks:

- **Permanent Water Mask:** Using the JRC Global Surface Water dataset to remove lakes/rivers that are always there.
- **Slope Mask:** Using HydroSHEDS DEM to remove steep terrain (> 5 degrees) that mimics water backscatter.
- **Connectivity Mask:** Removing isolated noise pixels by enforcing a minimum cluster size of **17 connected pixels**.

Finally, the raster is converted to vector polygons using reduceToVectors, and we apply a final area filter to drop any polygon smaller than **1.5 sq km** to ensure we only report significant flooding.

7.2 Hazard-Aware Routing Engine (GraphHopper)

The routing engine utilizes **GraphHopper**, customized to support dynamic "blocking" of areas. The frontend logic plays a crucial role in preparing the request for the engine.

7.2.1 Graph Construction

GraphHopper builds a navigable graph in memory from OpenStreetMap data. We explicitly disabled Contraction Hierarchies (ch.disable: true) to allow for dynamic edge weighting at runtime.

7.2.2 Frontend Request Construction

The Live Data page in the React application manages the routing logic. When a user requests a route, the application:

1. **Aggregates Hazards:** It collects all active "Blocked Zones" from the state (zones). This includes satellite floods, crowdsourced reports, and manual drawings.
2. **Formats Payload:** It constructs a MultiPolygon GeoJSON object containing all these blocked areas.
3. **Sends Request:** It sends a POST request to the GraphHopper custom model endpoint.

```
// Constructing the Custom Model payload
body.custom_model = {
    areas: {
        hazard_zones: {
            type: "Feature",
            geometry: {
                type: "MultiPolygon",
                coordinates: multiPolygonCoordinates, // All active zones
            },
            properties: {},
        },
    },
    // Priority logic: Multiply weight by 0 (Strict Avoidance)
    priority: [{ if: "in_hazard_zones", multiply_by: 0 }],
};
```

7.2.3 Dynamic Avoidance Toggles

A key feature is the ability to toggle specific avoidance layers.

- **Auto-Avoid Floods:** Automatically generates buffer polygons (approx. 500m radius) around satellite-detected centroids.
- **Avoid Reports:** Automatically generates buffer polygons (approx. 250m radius) around user-submitted incident markers.

```
const handleAutoAvoidToggle = (active) => {
    if (active) {
        // Generate 500m buffer zones around flood points
        const newZones = floodEvents.map(event => ({
            // ... create polygon logic
            source: "flood",
            isBlocked: true
        }));
        setZones(prev => [...prev, ...newZones]);
    } else {
        // Filter out flood zones
        setZones(prev => prev.filter(z => z.source !== "flood"));
    }
};
```

7.3 Frontend Development (React & Leaflet)

The user interface is built using **React.js** with **TypeScript** for type safety. The geospatial visualization is handled by **Leaflet.js**, managed via useRef hooks for direct map manipulation.

7.3.1 Map Initialization and Layer Management

We utilized useRef to maintain a reference to the Leaflet map instance (mapInstance) and specific layer groups (floodLayerRef, reportsLayerRef). This avoids re-rendering the entire map component when data updates, ensuring smooth performance.

- **Custom Icons:** We implemented L.divIcon with SVG strings to create lightweight, scalable markers for Rescue Centers (Red Cross) and Incidents (Orange Warning).
- **Layer Toggling:** useEffect hooks monitor the showRescueCenters and showReports state variables, adding or removing the corresponding LayerGroups from the map instantly.

7.3.2 Manual Polygon Drawing

For authority-led manual overrides, we integrated the leaflet-draw library.

- **Drawing Tool:** When isDrawing is true, a L.Draw.Polygon tool is enabled.
- **Event Handling:** Upon the L.Draw.Event.CREATED event, the new polygon is added to the drawnItemsRef FeatureGroup, and a new "Zone" object is added to the React state with isBlocked: true.

```
map.on(L.Draw.Event.CREATED, (e) => {
    const layer = e.layer;
    drawnItemsRef.current.addLayer(layer);
    // Add to state for routing calculation
    setZones(prev => [...prev, { id: L.Util.stamp(layer), isBlocked: true }]);
});
```

7.3.3 Interaction Management

A critical implementation detail was managing click conflicts. When a user is drawing a polygon or setting a "Start/End" point, clicks on underlying GeoJSON layers (like state boundaries) must be ignored. We implemented a robust interaction manager that temporarily sets pointer-events: none on the state layers during drawing modes, ensuring the map receives the click, not the boundary layer.

7.4 System Integration

The disparate components are glued together via a robust API workflow. The Python script runs on Google Cloud servers to process satellite data and export CSVs. The React frontend consumes these CSVs/APIs to populate the map. When a route is requested, the frontend bundles the visual map state (drawn zones + flood markers) into a JSON payload and ships it to the GraphHopper server, which returns the optimized path. This tight integration ensures that what the user *sees* (red zones) is exactly what the routing engine *avoids*.

CHAPTER 8

RESULTS

8.1 Web Dashboard Landing Page

This landing page acts as the primary gateway for all users, fetching real-time operational metrics such as active volunteers, rescue centres, and incident reports directly from the NEON backend. By presenting an immediate, high-level snapshot of the crisis (as seen in the status cards), the system efficiently routes authorities, volunteers, and the public to their specific operational workflows.

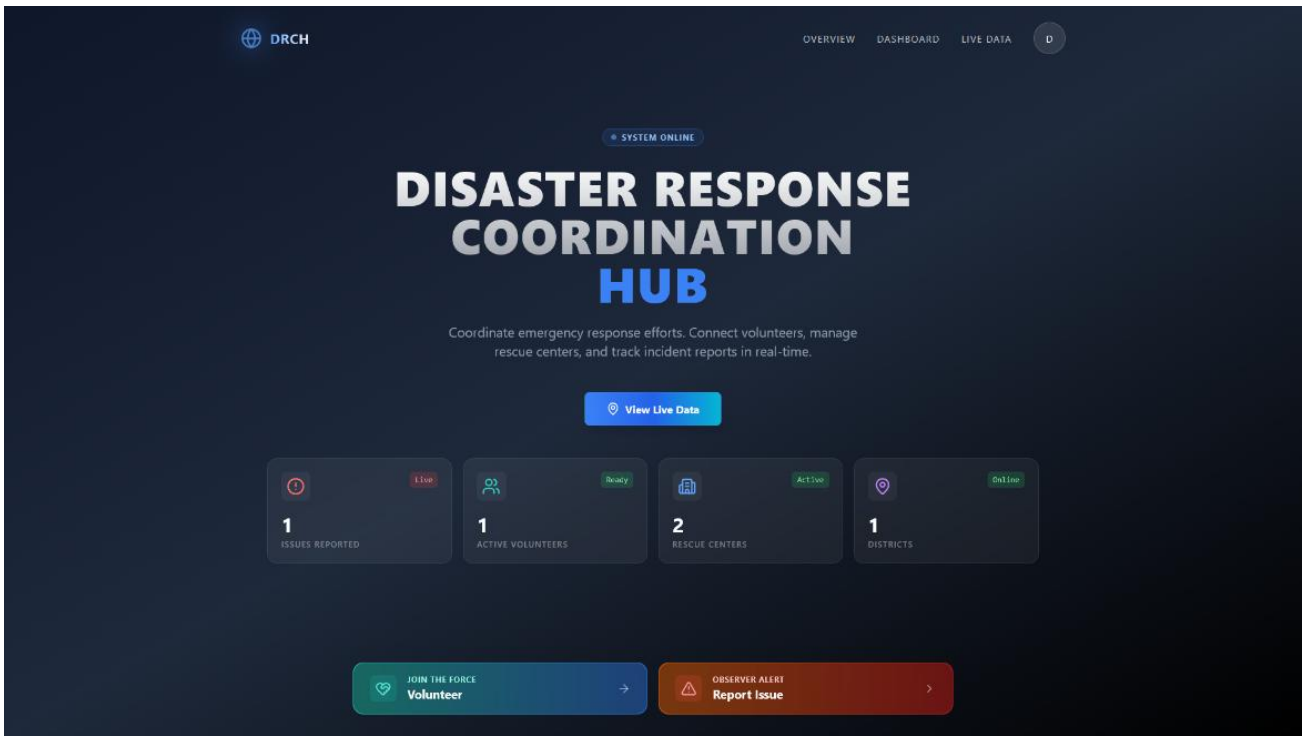

Figure-12: Web Dashboard

Figure 12 illustrates the central command dashboard of the Disaster Response Coordination Hub, developed using **React** and **Tailwind CSS** for a responsive user interface.

8.2 Authority Dashboard

This interface functions as the central command hub for administration, aggregating critical operational metrics—such as active incidents, available volunteers, and rescue center capacity—into a single, real-time view. It streamlines resource management by allowing authorities to verify incoming crowdsourced reports, assign specific tasks to field volunteers, and oversee the status of relief camps directly from a secure web portal, replacing chaotic manual coordination with a structured digital workflow.

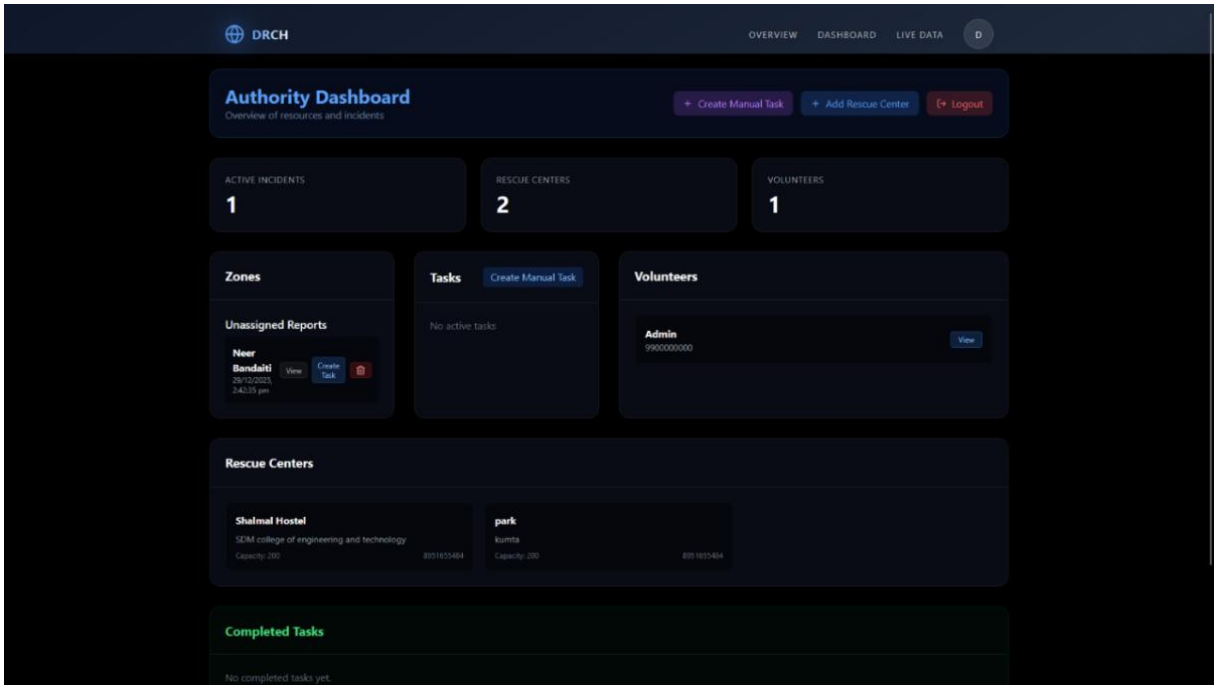

Figure-13: Authority Dashboard

Figure-13 presents this administrative view, specifically highlighting the "Unassigned Reports" section where incoming alerts await verification and the "Rescue Centers" panel for tracking shelter occupancy. By consolidating these disparate logistical elements into one cohesive dashboard, the system empowers authorities to maintain total situational awareness and execute rapid, evidence-based decisions during the heat of a disaster response.

8.3 Volunteer Task Dashboard

The Volunteer Dashboard serves as a focused tactical interface for field responders, streamlining complex command data into individual prioritized assignments. This personalized view replaces fragmented communication with a clear digital workflow, ensuring volunteers receive actionable instructions directly on their devices without being overwhelmed by system-wide data.

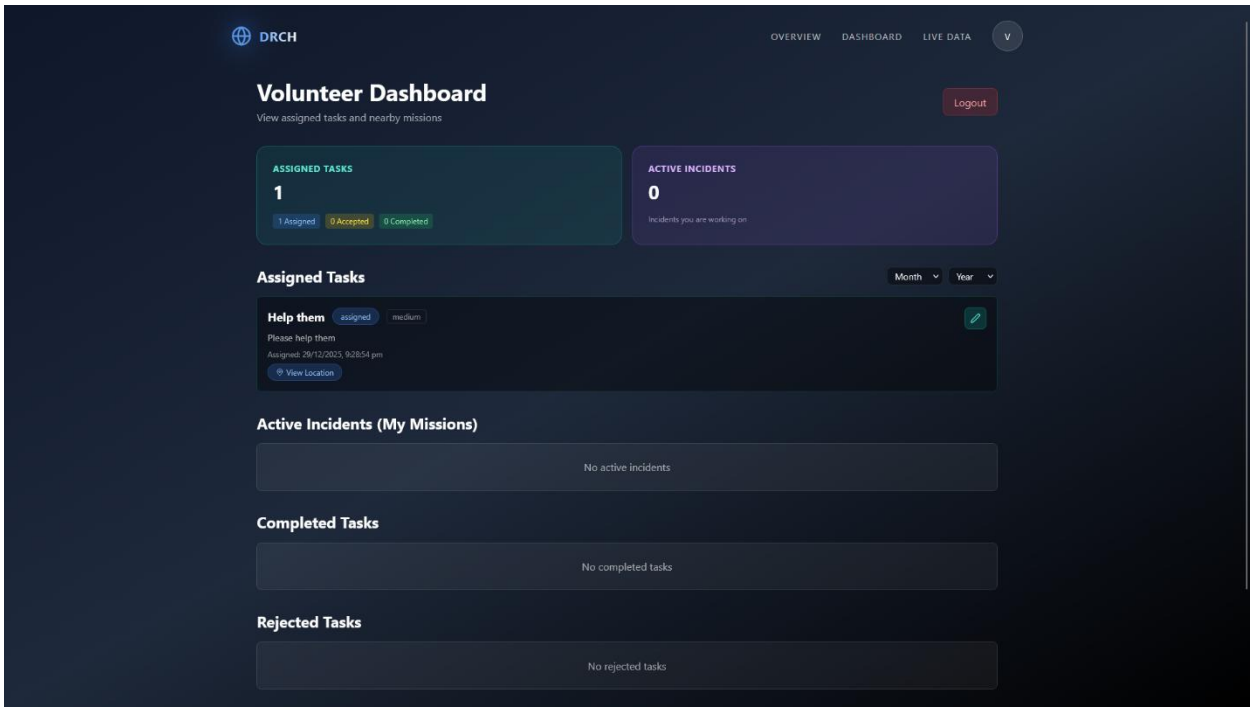

Figure-14: Volunteer Task Dashboard

Figure-14 illustrates this interface, specifically highlighting the "Assigned Tasks" section where new missions appear with critical metadata like priority levels and timestamps. The inclusion of a "View Location" button and real-time status counters ensures rapid situational assessment and seamless progress tracking for the central control room.

8.4 Visualized Flood Zones

The figure depicts this critical hazard zones overlaid directly onto the map interface, serving as a verified footprint for operational decision-making. By rendering this data as clear visual overlays, the system provides authorities with immediate situational awareness and acts as the foundational layer for the hazard-aware routing engine to calculate safe paths avoiding these confirmed risks.

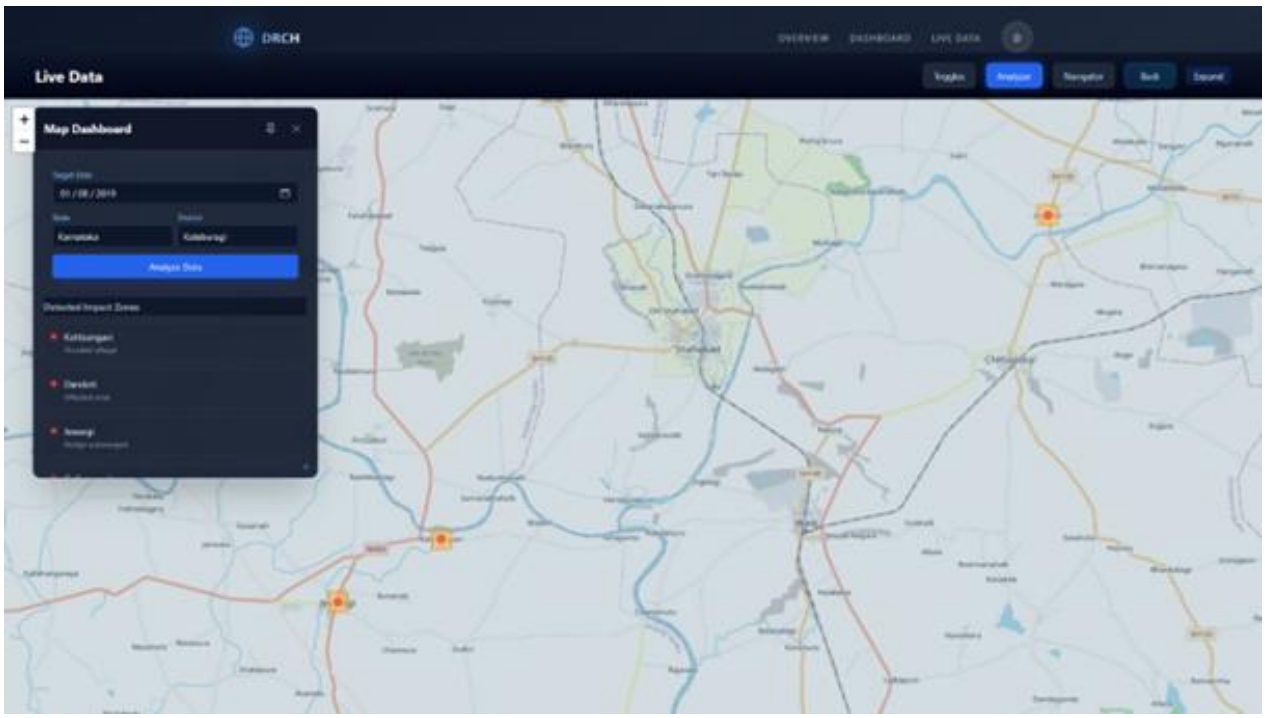

Figure-15. Flood Area (Zones Populated By GEE)

Figure-15 illustrates the August 2019 Kalaburagi flood event, where inundation zones appear as distinct red clusters with polygons along riverbanks. These polygons were generated by the system's automated pipeline using Sentinel-1 SAR imagery, which isolated significant changes in radar backscatters to accurately distinguish temporary floodwater from permanent water bodies and normal terrain.

8.5 Hazard-Aware Routing Interface

The routing engine integrates diverse hazard layers, allowing users to calculate safe paths by avoiding **Sentinel-1** floods, crowdsourced reports, or manually drawn zones. By toggling these constraints, the system finds the shortest route that strictly bypasses selected risks.

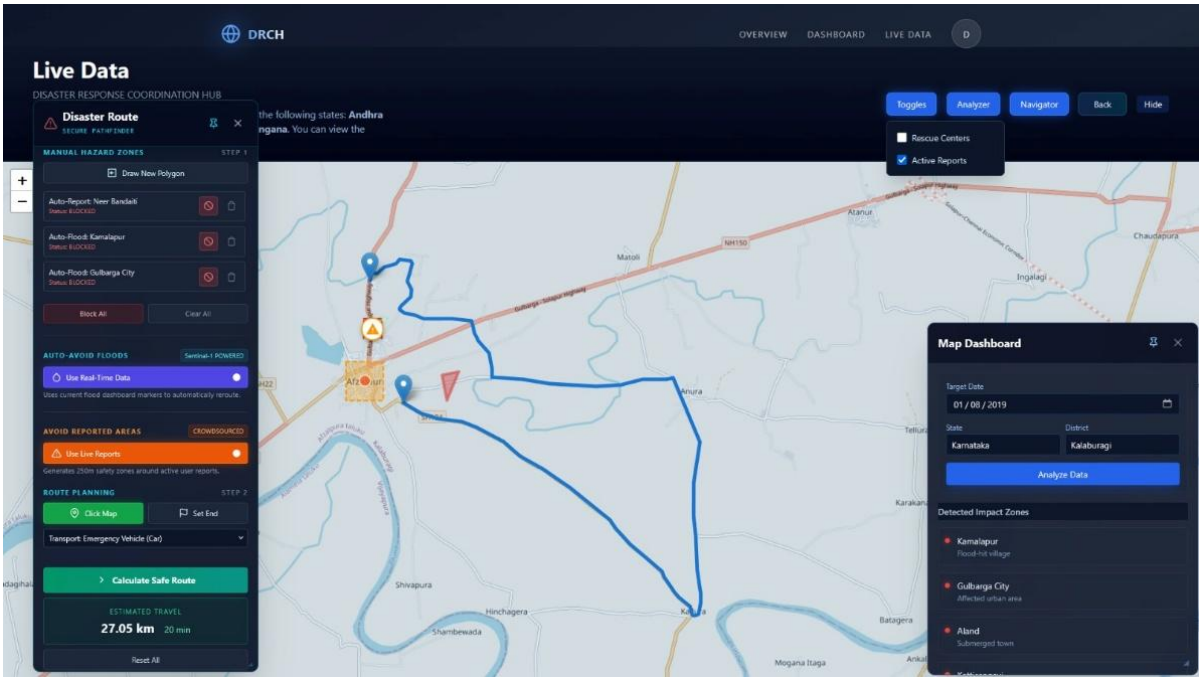

Figure-16: Routing After Avoiding Flood Area

Figure-16 demonstrates this capability in the "Live Data" view, where the blue navigation path actively deviates to avoid a specific hazard zone. The sidebar highlights the user's control, showing options to "Draw New Polygon" or enable "Auto-Avoid" for satellite and crowd data, ensuring that rescue teams can dynamically plan safe travel even as ground conditions change.

CHAPTER 9
CONCLUSION

The **Disaster Response Coordination Hub (DRCH)** project has successfully demonstrated that the gap between high-level satellite intelligence and on-the-ground disaster response can be bridged through a unified technological framework. By integrating **Sentinel-1 Synthetic Aperture Radar (SAR)** imagery, **Google Earth Engine (GEE)** processing pipelines, and **GraphHopper's** dynamic routing algorithms, the system provides a comprehensive solution for managing the chaotic logistics of flood relief operations.

Summary of Achievements The project achieved its primary objective of automating the flood detection-to-action workflow. We successfully implemented a pipeline that ingests raw satellite data, applies advanced speckle filtering (Gamma Map) to remove noise, and converts inundation masks into vector polygons. Crucially, these polygons are not merely static visual layers; the integration with the routing engine proves that "hazard-aware navigation" is technically feasible and operationally valuable. The system effectively calculates safe paths that actively avoid detected floodwaters, a capability that standard navigation apps currently lack.

Furthermore, the centralized dashboard has established a robust command-and-control structure. By enabling authorities to verify crowdsourced reports and assign tasks to registered volunteers, the platform transforms the disaster response from a reactive, disjointed effort into a coordinated, data-driven operation. The use of **Neon DB** has provided a scalable, serverless backend capable of handling real-time data synchronization across multiple users.

Addressing the Operational Gap Existing systems, such as the Central Water Commission's forecasting models or public crisis maps, operate in isolation—providing either broad warnings or unverified public alerts. The DRCH addresses the critical "last mile" information void by connecting these domains. It moves beyond the *observation* of disasters to the active *management* of response, ensuring that the rigorous data derived from remote sensing directly informs the practical decisions made by drivers and rescue teams on the ground.

Limitations While the system establishes a strong foundation, it is subject to certain limitations inherent to the current technology stack:

- **Satellite Revisit Time:** The system relies on Sentinel-1, which has a revisit cycle of 6–12 days. This means the satellite-derived flood polygons may not always reflect the absolute real-time status of a rapidly evolving flash flood.
- **Connectivity:** The reliance on a cloud-based architecture requires stable internet connectivity. While the interface is lightweight, deep routing functionality may be compromised in areas with total network blackouts.
- **Data Verification:** While the authority verification workflow reduces noise, the system ultimately relies on the quality of crowdsourced data for hazards that the satellite cannot see (e.g., urban drainage overflow).

Future Scope The modular architecture of the DRCH allows for significant future enhancements. Potential extensions include:

- **Drone Integration:** Incorporating real-time imagery from UAVs to update flood polygons in between satellite passes.
- **Offline Navigation:** Developing a fully offline mobile module that caches the road graph and flood layers, allowing volunteers to navigate without active internet.
- **AI-Driven Tasking:** Implementing machine learning algorithms to automatically match volunteer skills (e.g., medical training) with specific incident requirements.

In conclusion, the Disaster Response Coordination Hub serves as a scalable prototype for the future of disaster management. It validates that with the right integration of remote sensing and software engineering; we can build resilient systems that not only map disasters but actively guide communities to safety.

REFERENCES

[1]. Central Water Commission (CWC), "Flood Forecasting Network and Operational Guidelines," Government of India, 2020.

[2]. National Disaster Management Authority (NDMA), "National Disaster Management Guidelines: Management of Floods," Government of India, 2017.

[3]. Google Crisis Response, "Google Crisis Map," Google, Accessed: 2024.

[4]. S. Martinis, "Flood Mapping with Sentinel-1 SAR: Techniques and Applications," *Remote Sensing*, vol. 12, no. 11, pp. 1–22, 2020.

[5]. H. Shen et al., "SAR-Based Flood Detection Using Change Detection and Thresholding," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 58, no. 4, pp. 2645–2657, 2020.