1. Problem Tackled

Disaster response operations face critical challenges in speed, coordination, and scalability. Traditional methods rely heavily on manual workflows that are fragmented across stakeholders, making it difficult to triage help requests, match them with available resources, and respond effectively during surges.

ResQConnect addresses these inefficiencies by introducing a unified, intelligent system that:

- Enabling multimodal intake (text, image, voice) with automatic geotagging.
- Prioritizing requests via AI based on urgency, location, and resource availability.
- Coordinates task assignments and resource allocation across **four user roles**: Affected Individuals, Volunteers, First Responders, and Government Coordinators/Admins.
- We maintain human-in-the-loop oversight alongside ethical guardrails to ensure AI remains controlled, reliable, and aligned with responsible use.

2. AI Technologies & Concepts Used

Functionality	**Description**	**Technologies/ Concepts**
NLP & Multimodal Understanding	Extracts need type, urgency, and location from user inputs such as text, images, and voice.	Langgraph, Groq
Retrieval-Augmented Generation (RAG)	Uses a FAISS vector store of disaster-specific documents to generate context-aware task recommendations.	FAISS, RAG
AI-Powered Chat & Multiagent Interaction	Employs a 9B-parameter LLM with conversation summarization to maintain context and respond using chat history and disaster databases.	MCP CLient Server, Gemma2 (9B LLM)
Multi-Agent Framework & Human-in-the-Loop	Separates responsibilities across agents; all AI suggestions are routed through an admin approval step.	-
Observability & Feedback Integration	Instruments agent calls and chat interactions in Langfuse for real-time cost tracking and performance monitoring.	Langfuse, MCP

3. Architecture & Agent Workflow

3.1 Software Architecture: Software Architecture Diagram.png

3.1.1 Frontend

- Lightweight, modular **Progressive Web Application** with **IndexedDB** and **Service Workers** for offline caching and sync of data.
- Responsive **Next.js** interface.

3.1.2 Backend & Authentication:

- **FastAPI** backend handles all requests.
- Enforces **Firebase JWT authentication** and **role-based access control**.

3.1.3 Async & AI Tasks:

- **Celery workers** manage long-running tasks like AI agentic workflows.
- **Redis** acts as both a **message broker** and **hot cache**.

3.1.4 Storage & Data:

- Media uploaded directly to **ImageKit Storage Bucket**, bypassing the backend.
- Core data stored in **Firestore** with optimized schema and indexes.

3.1.5 DevOps & Security:

- Backend is **Dockerized**, CI/CD via **GitHub Actions**, deployed to **PaaS** (Railway).
- Maintains **separation of concerns**, **branching standards**, and **Firebase security rules** for scoped access.

3.2 Agentic architecture: AI agent architecture.png

Agent	Tasks
3.2.1 Intake Agent	- Receives text, image or voice requests - Performs geotagging, reverse geocoding and metadata (need type, count estimates) extraction - Passes the new request state along the agentic workflow.
3.2.2 Disaster Agent	- Matches incoming requests to existing disaster IDs. - Suggests a new disaster event to the admin when no match is found.
3.2.3 Task Agent (RAG pipeline)	- Retrieves relevant disaster documents from FAISS index - Generates step-by-step instructions and assigns urgency. - Writes recommended tasks to Firestore database and creates a ‘new task’ event.
3.2.4 Allocation Agent	- Queries Firestore for available volunteers and resources by disaster ID. - Runs a proximity-aware optimization to assign volunteers/ resources and recommends them to admins.

3.2.5 Orchestrator Agent	- Routes callbacks and status updates back to the API Gateway for client notifications. - Streams metrics and logs to Langfuse for observability.
3.2.6 Chatbot & Communication Hub	- Maintains full Text and in-app chat history as context. - Uses Gemma2 to summarize long conversations, preserving tokens. - Has read-only access to Firestore (via the DB MCP Server) for real-time context and logs interactions to Langfuse. - Also communicates with a RAG based MCP server to give more domain aware answers.
3.2.7 Data Collection Agent	- Runs scheduled cron jobs to fetch external news/SOP updates via a News API. - Acts as a serverless endpoint for up to date data collection. .

4. Implementation Highlights

- **Reliability & Scalability: Celery + Redis** guarantees no background task is lost- even if the API server restarts and allows horizontal scaling of workers.
- **Modular Codebase & Separation of Concerns:** Distinct python packages for authentication, request ingestion, AI agent orchestration, and resource management enabled rapid development.
- **User satisfaction** to evaluate chatbot responses is assessed through integrated thumbs up/down feedback mechanisms within the chat interface.
- **Security & Access Control**
    - **Firebase Authentication + Role-Based Access Control**: Fine-grained permissions for four user roles.
    - **Image Storage**: Direct-to-bucket uploads generate shareable links, bypassing backend overhead.
- **Performance Optimizations**
    - **Database Schema**: Normalized for minimal query count
    - **Frontend Caching**: Reduces API hits and improves load times

5. Challenges Faced

1. Ensuring agent suggestions were reliable while avoiding “automation complacency” required careful UX design for admin approvals.
2. Combining text, image, and location data into a unified vector for RAG retrieval involved custom preprocessing pipelines.
3. Building a custom allocation algorithm that minimized API usage without sacrificing recommendation quality.
4. Maintaining low-latency updates across many concurrent users and tasks drove optimization in both Celery worker tuning and Redis configuration.