---

1. Project Overview

Use Case Title:
WhatsApp Group Summarization with Role-Based Rule-Driven Insights

Team Name:
Team 48

Team Members & Roles:

- **Omkar** – LLM & Backend Developer
- **[Member 2 Name]** – Data Engineering & MongoDB
- **[Member 3 Name]** – Prompt & Rule Engineering
- **[Member 4 Name]** – Frontend & Integration

Problem Statement (in your own words):
Police officers manage high-volume coordination in WhatsApp groups. Manually tracking action points, escalations, and individual responsibilities across hundreds of messages is inefficient and error-prone. This creates bottlenecks in communication and impacts timely field action.

Solution Summary:
Our system summarizes WhatsApp group chats by officer role, extracting tasks and key updates using LLMs guided by hierarchical rule-based prompts. It maintains context using vector embeddings, supports multi-group and multi-officer queries, and displays personalized summaries through a FastAPI backend and React frontend.

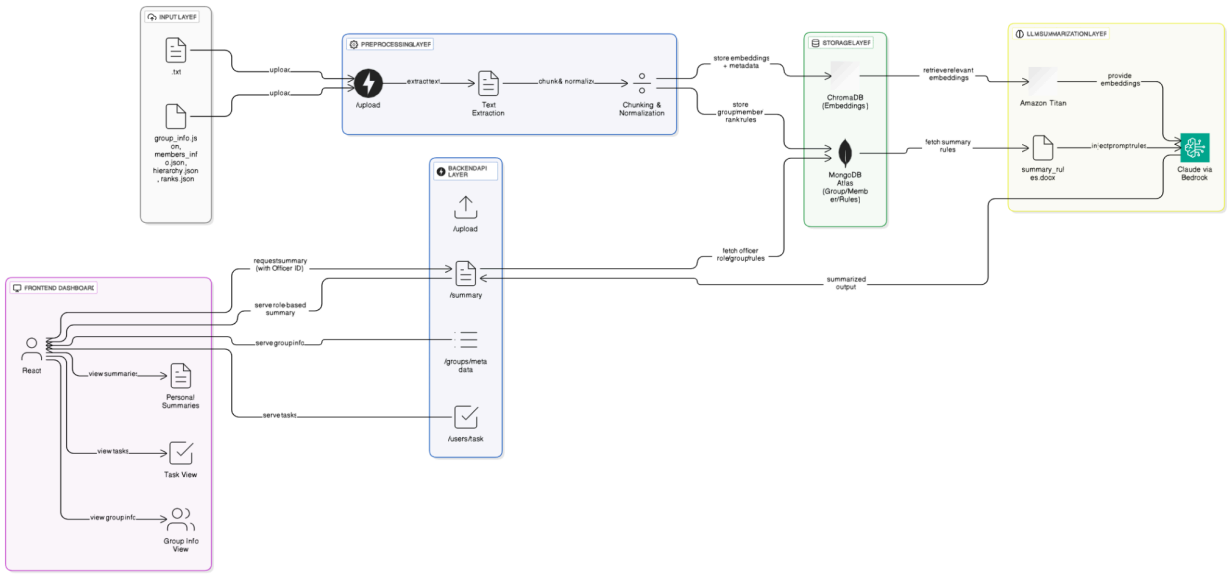
⚙️ 2. Technology Stack & Selection Rationale

2.1. Tech Stack Summary

Layer	Tool/Tech Used	Options Considered	Final Choice Justification
LLM/Model	Claude	e.g., Claude, Mistral	Accuracy, latency, ease of use, open vs closed model
Database	ChromaDB	e.g., PostgreSQL, Redis	NoSQL fit, speed, ease of integration
Backend/API	FastAPI	e.g., Flask, Node.js	Async support, speed, Python-native
Frontend	React	e.g., Vue, Angular	Ecosystem maturity, component reuse
Orchestration	LangChain	e.g., Haystack, Semantic Kernel	Prompt chaining, agent support
Deployment	null	e.g., Cloud Run, EC2, Vercel	Portability, free-tier limits, speed

---

🧩 3. Architecture Diagram & DATAata Flow



eraser

3.1 System Architecture:

- **Input:** WhatsApp chat `.txt` files, metadata JSONs (`group_info.json`, `ranks.json`, etc.)
- **Processing:** Files are parsed, chunked, embedded using Amazon Titan, and stored in ChromaDB
- **LLM Summarization:** Claude generates summaries using structured role-based prompts
- **Storage:** MongoDB stores officer metadata, rank, and group mapping
- **Frontend/API:** FastAPI exposes APIs, React renders officer-specific insights

3.2 Data Flow:

- Upload → Extract & Chunk → Embed (Titan) → Store (ChromaDB & Mongo) → Role-specific Prompt Injection → Claude Summary → API Output
- **Context Maintenance:** Chunks are stored with metadata, linked via role/group mapping

- **Decision Points:** Rules filter inputs → top-k search → LLM prompt decision

🛠️ 4. Configuration & Environment Setup

4.1 Configuration Parameters

- **LLM Keys:** Stored in `.env` file using `python-dotenv`
- **Thresholds:** Cosine similarity for vector search (≥ 0.75)
- **Paths & Secrets:** Mongo URI, ChromaDB path, AWS access keys

4.2 How to Run the Solution

1. Clone repository and navigate to `backend/`
2. Run `python -m venv venv` and activate it
3. Install requirements: `pip install -r requirements.txt`
4. Launch: `uvicorn main:app --reload`

API Endpoints:

- `/upload` [POST] – Upload chat + metadata
- `/groups/metadata` [GET] – Get group names & IDs
- `/groups/userid/task` [POST] – Tasks per officer
- `/users/{uid}/groups/{gid}/summary` [GET] – Role-aware summaries
- `/admin/reset-memory` [POST] – Clear ChromaDB
- `/admin/wipe-disk` [POST] – Delete storage folder
- ---

🧱 5. Component Inventory & Documentation

5.1 Functions / Scripts

Function Name	Purpose	Input	Output	Tech Used
`upload_files()`	Handles uploads, parsing & vectorization	Form files	File + chunk status	FastAPI, Python
`add_to_vectorstore()`	Stores chunk embeddings into ChromaDB	Chunk + metadata	Stored embedding	ChromaDB, Titan
`user_task()`	Fetch officer-level task summaries	phone number	JSON summary	Claude, Mongo

5.2 APIs

API Name	Method	Endpoint	Purpose	Request Format	Response Format
Upload Files	POST	`/upload`	Upload WhatsApp + JSON files	Form	JSON

Group Metadata	GET	`/groups/metadata`	Fetch group IDs/names	-	JSON
Role Summmary	GET	`/users/{uid}/groups/{gid}/summary`	Officer summary	Query pararams	JSON

5.3 Prompts Used

Prompt Name	Use Case	Input Type	LLM Output	Versioned?
Summary Prompt	Role-based summary	Role + chunks	Bullet-point summary	✅
Task Extractor	Task identification	Chat log	Structured task report	✅

5.4 Agents (if used):

Not applicable – rule-based routing with prompt templates used.

5.5 Other Components

- **Embedding Models:** Amazon Titan
- **Vector Stores:** ChromaDB

- **Search Pipelines:** Cosine similarity + top-k retrieval
- **Pre/Post-Processing:** File parsing, chunking, role filtering, JSON formatting
-

6. Known Limitations

- LLM may over-summarize or miss context if input quality is low
- Non-English (Telugu) messages not well handled
- Rule precision depends on metadata completeness
- Summaries must be verified for high-stakes use cases

📈 7. Improvement Areas & Next Steps

- Add OCR for image-based chat logs
- Build dynamic UI filters by officer role/jurisdiction
- Fine-tune LLM on police-specific chat summaries
- Modularize into Dockerized services: chunker, embedder, summarizer
- Enable secure deployment with AuthN/AuthZ

🗂️ 8. Supporting Files

- Architecture Diagrams (Eraser-generated, PNG)
- Prompt Files: `summary_rules.docx`, `Role Specific rules_Sample.txt`
- `.env` template for configuration

- Sample WhatsApp `.txt` and JSON hierarchy files
- Example summary output per officer/role
