Solution Approach Documentation: Self-Healing Test Tool

1. Problem Statement

Automation testing failures often occur due to UI changes or unexpected exceptions. This self-healing test tool aims to automatically detect, diagnose, and fix these failures, ensuring test scripts remain robust and up to date without manual intervention.

2. Given Input and Output

Input:

- GitHub repository or file

Output:

- Updated repository or file with fixed test scripts

3. Solution Overview

A **multi-agent AI pipeline** will handle the self-healing process. The agents will share memory, follow structured state management, and communicate via a **centralized chat manager**.

Core Technologies & Frameworks:

- **Swarm AI** → Multi-agentic AI framework
- **Neo4j** → Knowledge graph for repository traversal and backpropagation
- **ChromaDB** → Vector memory for shared context
- **GROQ API (LLAMA 3.3 70B)** → LLM model for reasoning and decision-making
- **Orchestrator Workflow** → Task assignment and dependency management
- **Compression Techniques** → Manage growing context size
- **LLM Reflection & ReAct Techniques** → Enhance self-learning and adaptability

4. Multi-Agent Architecture

Agent Name	Functionality
Finder	Identifies test failures using appropriate tools (e.g., Playwright, Selenium logs, test reports).
Detector	Diagnoses where the failure occurs and checks dependencies (GitHub traversal, knowledge graph traversal).
Investigator	Investigates the root cause of failure (UI changes, broken selectors, API changes).
Solution Provider	Generates potential fixes based on previous agents' context (LLM reasoning + RAG).
Solution Applier	Applies the fix to the repository (GitHub API, script modification tools).
Solution Tester	Validates the applied fix by re-running test cases.
Chat Manager	Monitors agents, resolves conflicts, and ensures structured communication.

5. Workflow Execution

1. **GitHub Repository Conversion** → Convert the repo into a **knowledge graph** (Neo4j).
2. **Failure Detection** → Finder agent scans logs, test reports, and exception messages.
3. **Root Cause Analysis** → Detector and Investigator agents analyze the failure, dependencies, and test environment.
4. **Solution Proposal** → Solution Provider suggests fixes using **RAG + LLM reflection techniques**.
5. **Solution Application** → Solution Applier updates scripts using **GitHub API** or **code modification tools**.
6. **Testing & Validation** → Solution Tester executes modified scripts, ensuring correctness.
7. **Backpropagation Loop** → If a failure persists, the pipeline loops back to earlier agents.

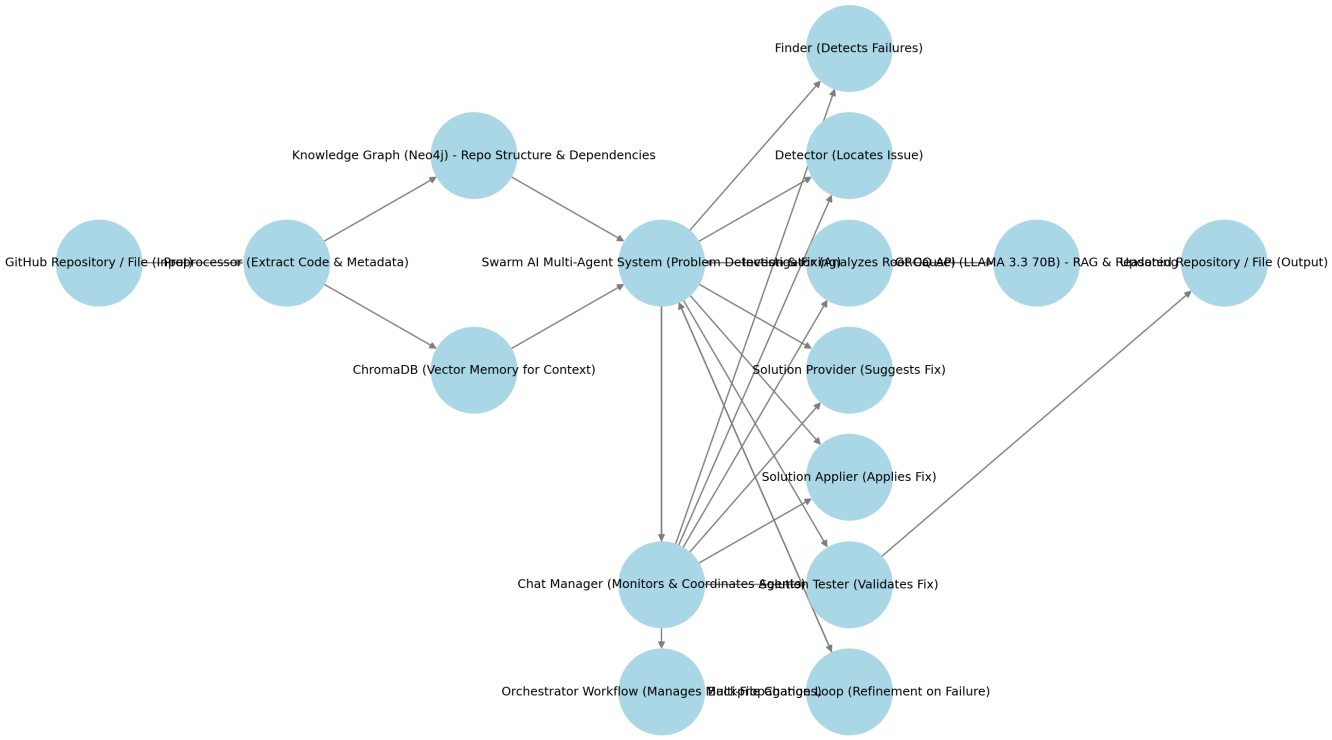

Fig : Architecture Diagram

6. Tools & Prompting Strategy

Agent	Tool Used	Prompt Strategy
Finder	Selenium, Playwright logs, Allure reports	"Analyze the test logs and identify failed test cases along with error messages."
Detector	GitHub API, Neo4j traversal	"Locate the failure point in the test suite and check if changes in UI/API are the cause."
Investigator	Git diff, DOM comparison tools	"Analyze the failure cause - is it a selector issue, API response change, or missing element?"
Solution Provider	LLM (GROQ API) with RAG	"Generate a fix considering the test context and UI/API changes."
Solution Applier	GitHub API, Code modification tools	"Apply the following changes in the test script to resolve the issue."
Solution Tester	Selenium, Pytest, CI/CD pipeline	"Run the modified tests and verify success."

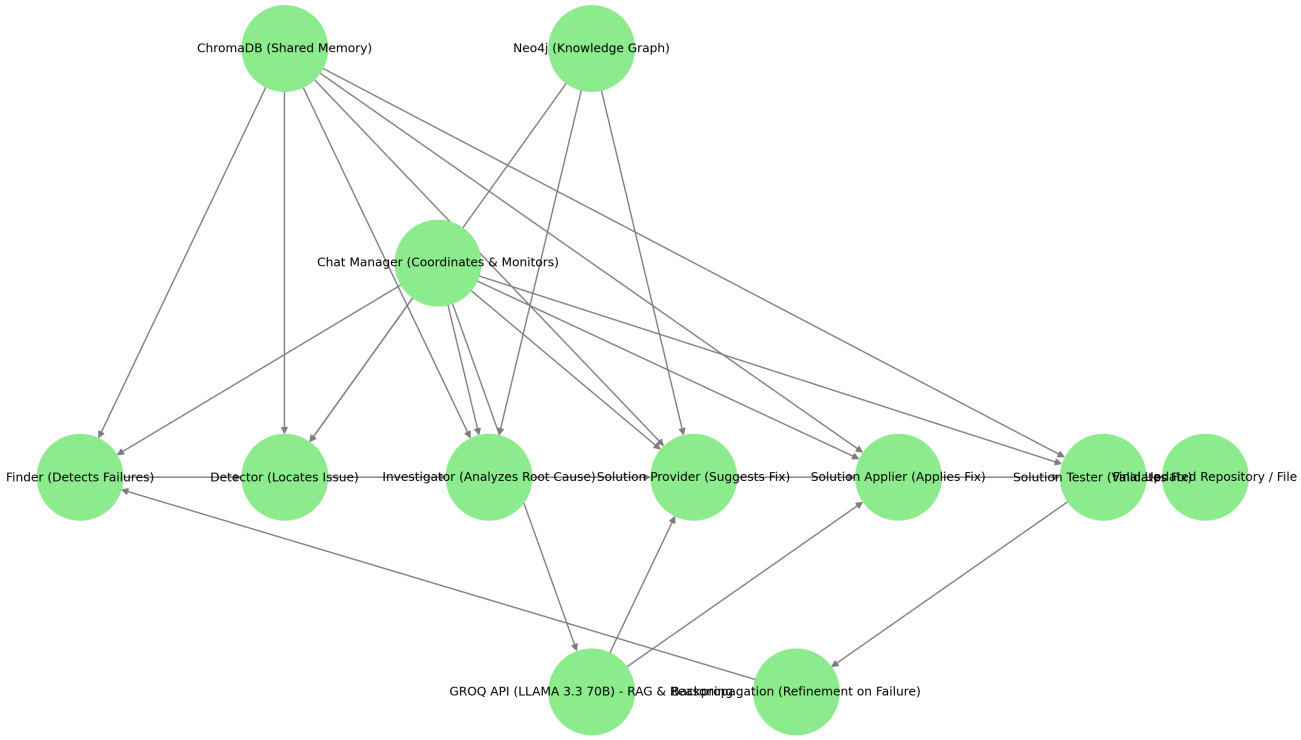

Fig : Agentic Workflow

7. State Management & Context Handling

- **Shared Chat & Memory:** All agents share a **global state** using ChromaDB for efficient information flow.
- **Structured State Management:** Agents work in sequential steps, with **Chat Manager** ensuring synchronization.
- **Compression for Long Contexts:** Summarization & pruning techniques will help in keeping prompts concise.

8. Implementation Considerations

- **Using Neo4j for Knowledge Graph** → Helps track **dependencies between scripts, UI elements, APIs**.
- **ReAct & LLM Reflection** → Agents use reasoning & prompting to iteratively improve solutions.
- **Global & Agent-Based Rewards** → Ensures learning-based optimization of fixes.
- **Swarm-Based Agent Coordination** → Efficient multi-agent collaboration via structured workflows.

9. Conclusion

This self-healing test tool will minimize human intervention in fixing automation test failures. By leveraging **multi-agent AI**, **knowledge graphs**, and **GROQ's LLM**, the system can **detect, diagnose, and resolve issues dynamically**. The **centralized state management and chat** ensure smooth collaboration, and **backpropagation** refines solutions when needed.