Emergency Response: A Multi-Agent System

Sheena Maria Lang, Antonio Lobo Santos, Zachary Parent,
María del Carmen Ramírez Trujillo and Bruno Sánchez Gómez

January 10, 2025

Contents

1 Introduction

This report presents the final implementation and results of our multi-agent system (MAS) for emergency response coordination. Building upon our previous designs from Tasks 1 and 2, we have developed a complete, functional system that demonstrates the effectiveness of agent-based approaches in managing complex emergency scenarios.

The system is implemented using CrewAI, a framework that enables the creation and coordination of specialized agent crews. Each crew is designed with specific responsibilities and operates through well-defined processes, ensuring efficient handling of emergency situations. The implementation includes:

- **Emergency Services Crew:** Handles initial emergency assessment and coordination
- **Firefighter Agent Crew:** Manages firefighting resources and operations
- **Medical Services Crew:** Coordinates medical response and hospital resources
- **Public Communication Crew:** Manages public information and communication

Report Structure:

- Section 2 details the design and implementation of each crew, including their process definitions and data models
- Section 3 explains the interaction mechanisms between crews and the overall system flow
- Section 4 presents the results of system testing and validation
- Section 5 concludes with insights and potential future improvements

The implementation builds upon our previous design while introducing several refinements based on practical considerations and testing results. These modifications are documented and justified throughout the report. The complete source code, along with setup instructions and required input files, is provided in the accompanying project repository.

2 Crew Design and Implementation

2.1 General Design Principles

For each crew in our system, a corresponding Python file is used to instantiate the configuration. These configurations are structured based on CrewAI's YAML schema recommendations for crews[1], tasks[2], and agents[3].

These files were generated using CrewAI CLI, a universal tool for creating configurations. The following command demonstrates how to initialize a new crew configuration:
`crewai create crew my_new_crew`[4].

The Python file is structured to include the following key elements:

- **Imports**: Required modules, including CrewAI components such as 'Agent', 'Task', 'Crew', and 'Process'.
- **Tool Instantiation**: Creation of tools for task-specific functionalities.
- **Agent Configuration**: Agents are defined with specific properties, including their roles, goals, delegation capabilities, verbosity, and parameters for interacting with the language model (e.g., temperature settings).
- **Task Configuration**: Tasks are defined with descriptions, expected outputs, dependencies, and execution modes. These configurations ensure tasks are properly structured and validated. Last task of each crew includes `output_pydantic` to ensure that the dictionaries returned to the crew are consistent.
- **Schema Augmentation**: Pydantic schemas can be added to the expected output of tasks using utility functions such as `add_schema_to_task_config`. This function modifies the task configuration by appending the schema JSON to the expected output property, ensuring that the LLM can take it into account.
- **Crew Composition**: The crew integrates agents and tasks into a defined process, executed sequentially, to accomplish its objectives.

The YAML configurations for agents and tasks specify several general properties:

[1] `https://docs.crewai.com/concepts/crews#yaml-configuration-recommended`
[2] `https://docs.crewai.com/concepts/tasks#yaml-configuration-recommended`
[3] `https://docs.crewai.com/concepts/agents#yaml-configuration-recommended`
[4] `https://docs.crewai.com/concepts/cli#1-create`

Agent Configuration: Agents are defined using YAML properties to specify:

- **Role:** The role the agent plays within the crew.
- **Goal:** The overarching objective or mission assigned to the agent.
- **Delegation Capabilities:** Whether the agent is allowed to delegate tasks.
- **Language Model Parameters:** Specific settings such as the model used and the temperature to control the randomness of the output.

Task Configuration: Tasks are defined in YAML with properties that include:

- **Description:** A clear explanation of the purpose and workflow of the task.
- **Expected Output:** The structure and format of the task output, often validated against a schema.
- **Dependencies:** Other tasks that provide context for the task.
- **Execution Mode:** Specifies whether the task is executed synchronously or asynchronously (e.g., 'async_execution: true').

This modular and schema-driven approach ensures flexibility, reusability, and validation throughout the configuration process.

A design overview of the system.

2.2 Tools

In this section, we describe the various tools developed for the Emergency Planner system. These tools are designed to facilitate different aspects of emergency management, including calculating route distances and managing database entries related to hospitals and incidents. Each tool is implemented with specific functionalities to address different requirements in emergency scenarios.

2.2.1 Route Distance Tool

Purpose The Route Distance Tool calculates the driving route distance between an origin and a destination based on their coordinates. This is essential for determining the quickest routes for emergency response teams.

Implementation

- **Input:** The tool requires the x and y coordinates of both the origin and destination locations.
- **Execution:** The city map graph is loaded from a GraphML file, and the shortest path is calculated using the travel time as the weight. The total distance is then computed.
- **Output:** The tool returns the total driving distance in kilometers.

The Route Distance Tool is a critical component to bring the Emergency Planner system closer to the real world. It gives the agents access to accurate geographical information, enabling them to make informed decisions about resource allocation and response times.

2.2.2 Database Management Tools

Purpose The Database Management Tools include the Hospital Reader, Hospital Updater, and Incident Retrieval tools. These tools manage and update the database entries related to hospitals and incidents, ensuring that the information is current and accurate.

Implementation

- **Hospital Reader Tool:**
  - **Input:** No input parameters are required for this tool.
  - **Execution:** The tool connects to the SQLite database and executes a query to fetch all hospital records.
  - **Output:** The tool returns a list of hospitals, including their IDs, locations, and available resources.
- **Hospital Updater Tool:**

  - **Input:** The tool requires the hospital ID, number of beds reserved, number of ambulances dispatched, and number of paramedics deployed.
  - **Execution:** The tool connects to the SQLite database and executes an update query to modify the hospital's available resources.
  - **Output:** The tool returns a confirmation of the update operation.

- **Incident Retrieval Tool:**
  - **Input:** The tool requires the x and y coordinates of the location, fire severity, fire type, and a summary of the new incident.
  - **Execution:** The tool connects to the SQLite database, retrieves related incidents based on proximity, fire severity, and fire type, and inserts the new incident into the database.
  - **Output:** The tool returns a list of related incidents.

The Database Management Tools are essential for maintaining up-to-date and accurate information on hospitals and incidents. By efficiently managing and updating the database, these tools ensure that the data is consistent throughout crews and runs, thus helping mitigate the hallucinative nature of the LLM-based agents.

2.3 Emergency Services Crew

2.3.1 Design

Purpose The Emergency Services crew is responsible for the initial assessment of the emergency situation and the dispatch of the appropriate response teams.

Changes In the initial design, the emergency services crew was responsible for directly notifying the other crews and generating separate assessments for the firefighters crew and the medical services crew, but we chose to instead generate a single call assessment, which includes the evaluation of whether firefighters and medical services are required. This assessment is then programmetically used by the flow to call the subsequent crews.

2.3.2 Implementation

Emergency Call Agent An explanation of the code.

```
demo code
```

Notification Agent An explanation of the code.

```
demo code
```