Emergency Response: A Multi-Agent System

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

January 12, 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 [2], 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, data models and tools
- Section 3 explains the interaction mechanisms between crews and the overall system flow
- Section 4 presents the design and 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

In this section, we will delve into the key design and implementation decisions for the construction of the Emergency Planner multi-agent system.

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.

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 using the OSMnx [1] package, 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 [3] 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.

To facilitate the utilization of these tools, a script named `sqlite-init.py`, located in `src/scripts/`, has been developed to initialize the SQLite database and populate its tables. Additionally, a Makefile is provided to generate a compressed archive of the project's source code, accessible by executing the `make zip` command.

2.3 Emergency Services Crew

2.3.1 Design

Purpose The Emergency Services Crew is tasked with the critical responsibility of assessing emergency situations and ensuring the appropriate response teams are dispatched effectively. This foundational role involves processing incoming emergency calls, structure the received information and decide which crews should be notified.

Changes In the initial design, the Emergency Services Crew was intended to utilize two specialized tools:

- *Database Management:* A system for systematically entering and recording emergency details.
- *GPS Mapping:* Software for locating and verifying the caller's position, assessing the scene, and prioritizing the severity of the incident if necessary.

However, after evaluation, it was determined that the core purpose of the crew was receiving the already transcribed incoming calls and deciding which team should receive the information, so both task could be accomplished without these tools. Removing them simplified the workflow, reduced overhead, and maintained focus on the essential tasks of the crew.

Structure and Components The Emergency Services Crew leverages the `CrewBase` framework to orchestrate its agents and tasks in a sequential process, ensuring efficient and orderly operations. The key components of the design are:

- **Agents:**
  - `emergency_call_agent`: Processes incoming emergency calls using a predefined configuration, ensuring the accurate extraction of relevant details.
  - `notification_agent`: Decides which crews need to be notified about the emergency, based on the nature and severity of the situation. This ensures the appropriate response teams are activated in a timely manner.

- **Tasks:**
  - `receive_call`: Captures and processes the details of emergency calls using schemas derived from the `EmergencyDetails` model, ensuring structured and consistent data handling.
  - `notify_other_crews`: Evaluates the information detailed in the `EmergencyDetails` and decides if other crews are needed, utilizing the `CallAssessment` schema to ensure clarity and comprehensiveness in the communication.

- **Crew:** The crew is constructed using the `Crew` class, integrating the agents and tasks into a seamless workflow. This design ensures the efficient progression from receiving an emergency call to notifying the relevant response teams.

The modular design of the Emergency Services Crew enables it to integrate seamlessly with other components of the multi-agent system while maintaining flexibility for updates or extensions.

2.3.2 Implementation

Emergency Call Agent The `emergency_call_agent` uses the configuration specified in `self.agents_config["emergency_call_agent"]` to ensure proper handling of call data. It acts as the first step in the emergency response workflow, extracting and structuring information from the calls.

```
@agent
def emergency_call_agent(self) -> Agent:
    return Agent(config=self.agents_config["emergency_call_agent"])
```

Notification Agent The `notification_agent` uses the configuration provided in `self.agents_config["notification_agent"]`. This agent ensures that only the relevant teams are alerted, streamlining the response process and avoiding unnecessary notifications.

```
@agent
def notification_agent(self) -> Agent:
    return Agent(config=self.agents_config["notification_agent"])
```

Receive Call Task This task processes the incoming call details and structures them according to the schema provided by the `EmergencyDetails` model. This ensures data consistency and validity.

```
@task
def receive_call(self) -> Task:
    config = add_schema_to_task_config(
        self.tasks_config["receive_call"], EmergencyDetails.model_json_schema()
    )
    return Task(config=config)
```

Notify Other Crews Task This task uses the processed emergency details to notify other relevant crews. It structures its output using the `CallAssessment` schema, ensuring clarity in communication.

```
@task
def notify_other_crews(self) -> Task:
    config = add_schema_to_task_config(
        self.tasks_config["notify_other_crews"], CallAssessment.model_json_schema()
    )
    return Task(config=config, output_pydantic=CallAssessment)
```

Crew Construction The agents and tasks are combined into a sequential process using the `Crew` class. This design ensures an orderly workflow from receiving calls to notifying the appropriate teams, where the Emergency Services Crew acts as a critical component for initial emergency assessment and coordination across multiple teams within the system.

```
@crew
def crew(self) -> Crew:
    """Creates the Emergency Services Crew"""
    return Crew(agents=self.agents, tasks=self.tasks, process=Process.sequential)
```

2.4 Firefighter Agent Crew

2.4.1 Design

Purpose The Firefighter Agent Crew is tasked with the critical responsibility of responding to fire emergencies by efficiently allocating resources, deploying personnel, and reporting on the actions taken. This crew ensures that firefighting efforts are conducted with precision, safety, and coordination.

Structure and Components The Firefighter Agent Crew leverages the `CrewBase` framework to orchestrate its agents and tasks in a sequential process, ensuring efficient and orderly operations. The key components of the design are:

- **Agents:**
  - `fire_chief`: Facilitates communication and coordination within the crew and with emergency service operators, ensuring the team operates effectively.
  - `equipment_technician`: Manages firefighting resources by determining, packing, and maintaining necessary materials for emergencies.
  - `firefighter`: Deploys to fire scenes, strategizes effective firefighting approaches, and extinguishes fires while ensuring safety.
- **Tasks:**
  - `allocate_firefighting_resources`: Determines the necessary firefighting materials based on the Fire Assessment, ensuring resources are ready and appropriate for the situation.
  - `deploy_fire_combatants`: Plans and executes the deployment of firefighters to the fire location, based on the Fire Assessment and allocated resources.
  - `report_firefighting_response`: Provides a comprehensive summary of the firefighting response and actions taken, ensuring a clear record of the operation.
- **Crew:** The crew is constructed using the `Crew` class, integrating the agents and tasks into a seamless workflow. This design ensures the efficient progression from resource allocation and deployment to comprehensive reporting of the firefighting response.

The modular design of the Firefighter Agent Crew enables it to operate as an independent yet integrative component within the broader multi-agent system, ensuring a flexible and efficient response to fire emergencies.

2.4.2 Implementation

Fire Chief Agent The `fire_chief` agent uses the configuration specified in `self.agents_config["fire_chief"]`. This agent facilitates communication and coordination within the Firefighter Agent Crew and ensures the team operates effectively during emergencies.

```
@agent
def fire_chief(self) -> Agent:
    return Agent(config=self.agents_config["fire_chief"])
```

Equipment Technician Agent The `equipment_technician` agent leverages the configuration from `self.agents_config["equipment_technician"]` to manage firefighting resources effectively by determining, packing, and maintaining necessary materials.

```
@agent
def equipment_technician(self) -> Agent:
    return Agent(config=self.agents_config["equipment_technician"])
```

Firefighter Agent The `firefighter` agent is configured through `self.agents_config["firefighter"]` to deploy to fire scenes, strategize effective firefighting approaches, and ensure safety during operations.

```
@agent
def firefighter(self) -> Agent:
    return Agent(config=self.agents_config["firefighter"])
```

Allocate Firefighting Resources Task This task determines the necessary firefighting materials based on the Fire Assessment, {`fire_assessment`}.

```
@task
def allocate_firefighting_resources(self) -> Task:
    config = add_schema_to_task_config(
        self.tasks_config["allocate_firefighting_resources"],
        FireResources.model_json_schema()
    )
    return Task(config=config, agent="equipment_technician")
```

Deploy Fire Combatants Task This task plans and executes the deployment of firefighters to the fire location, based on the Fire Assessment, {`fire_assessment`}, and the resource allocation.

```
@task
def deploy_fire_combatants(self) -> Task:
    config = add_schema_to_task_config(
        self.tasks_config["deploy_fire_combatants"],
        FireDeployment.model_json_schema()
    )
    return Task(config=config, agent="firefighter",
        context=["allocate_firefighting_resources"])
```

Report Firefighting Response Task The `report_firefighting_response` task provides a comprehensive summary of the firefighting response and actions taken.

```
@task
def report_firefighting_response(self) -> Task:
    config = add_schema_to_task_config(
        self.tasks_config["report_firefighting_response"],
        FireResponseReport.model_json_schema()
    )
    return Task(config=config, agent="fire_chief", context=["deploy_fire_combatants"])
```

Crew Construction The Firefighter Agent Crew is constructed as a sequential process to cohesively manage tasks and agents.

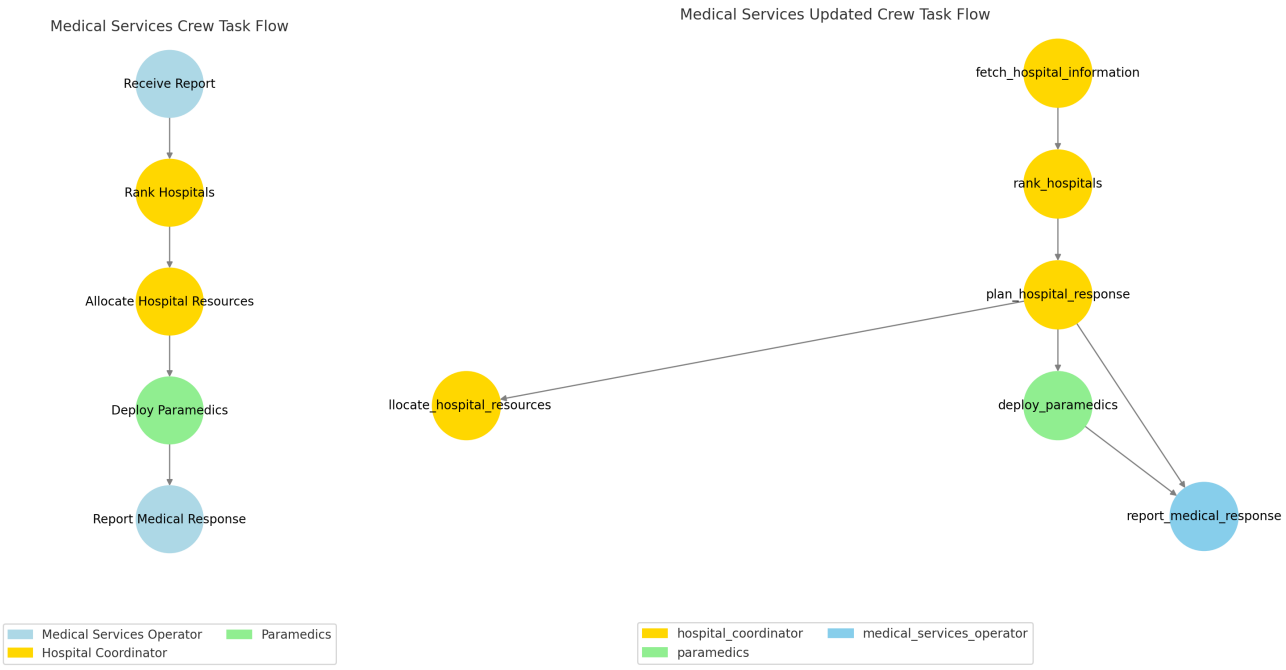
```
@crew
def crew(self) -> Crew:
    """Creates the Firefighter Agent Crew"""
    return Crew(
        agents=self.agents, tasks=self.tasks, process=Process.sequential, verbose=True
    )
```

2.5 Medical Services Crew

2.5.1 Design

Purpose The Medical Services Crew is responsible for managing and coordinating medical resources during emergency situations. This includes assessing hospital capacities, allocating resources, deploying paramedics, and ensuring timely medical response to incidents.

Changes Initially, the tasks of the Medical Services Crew were planned in a different order and with a distinct execution flow. A comparison of the initial and updated workflows is presented in **Figure** 1. In the updated process, the Recieve Report task was removed, since it was not necessary for the crew's operations. Additionally, the Rank Hospitals and Allocate Hospital Resources were each split into 2 separate tasks: Fetch Hospital Information and Rank Hospitals, and Plan Hospital Response and Allocate Hospital Resources, respectively. This restructuring was done to improve the efficiency and clarity of the crews' operations, since the tasks' results were poor when one agent had to perform too many steps at a time. We observed that the agents especially struggled when combining in a single task the use of a tool and the inference of logical results from said tool's output.



(a) Initial process

(b) Updated process

Figure 1: Comparison of the Medical Services Crew's execution flow. (a) shows the initial process, while (b) illustrates the updated process with task restructuring for improved efficiency.

Structure and Components The Medical Services Crew leverages the `CrewBase` framework to orchestrate its agents and tasks, ensuring cohesive and efficient operations. The key components of the design are:

- **Agents:**
  - `medical_services_operator`: Facilitates communication and coordination within the Medical Services crew and with other crews, ensuring efficient information flow and timely updates during emergencies.
  - `hospital_coordinator`: Manages all hospitals, decides each of their responses based on proximity to emergencies, and allocates resources effectively to ensure optimal response during crises.
  - `paramedics`: Plans the deployment of paramedics to the emergency site, including the number of paramedics, ambulances, estimated times of arrival, and any special equipment needed to handle the situation effectively.
- **Tasks:**
  - `fetch_hospital_information`: Uses the Hospital Reader Tool to fetch the information of all hospitals from the database and returns the hospital information in a structured output.
  - `rank_hospitals`: Receives input containing the Medical Assessment and the hospital information from the previous task, calculates the distance to the emergency location using the Route Distance Tool, and returns the list of hospital information with distances to the emergency.
  - `plan_hospital_response`: Reads the ranked list of hospitals and the Medical Assessment, assesses available resources at the given hospitals, and creates an allocation plan that details how resources will be utilized to address the emergency.
  - `allocate_hospital_resources`: Reads the Hospital Resource Allocation plan, uses the Hospital Updater Tool to update the database for all of the hospital resources used in the plan, and confirms that the hospital resources have been successfully allocated.
  - `deploy_paramedics`: Reads the Hospital Resource Allocation plan and the Medical Assessment, calculates the total number of paramedics to be deployed and ambulances to be dispatched, determines the estimated times of arrival, and identifies any special equipment required.
  - `report_medical_response`: Reads the Hospital Resource Allocation plan, the Paramedic Deployment plan, and the Medical Assessment, compiles a comprehensive summary of the medical response plan, and provides a detailed report for future reference and continuous improvement.

- **Crew:** The crew is constructed using the `Crew` class, which integrates the agents and tasks into a seamless workflow. This design ensures effective progression from hospital information retrieval and resource allocation to paramedic deployment and medical response reporting.

The modular design of the Medical Services Crew enables it to adapt to a wide range of emergency scenarios, ensuring efficient and timely medical response to incidents.

2.5.2 Implementation

Medical Services Operator Agent The `medical_services_operator` agent uses the configuration specified in `self.agents_config["medical_services_operator"]`. This agent facilitates communication and coordination within the Medical Services crew and with other crews, ensuring efficient information flow and timely updates during emergencies.

```
@agent
def medical_services_operator(self) -> Agent:
    return Agent(config=self.agents_config["medical_services_operator"])
```

Hospital Coordinator Agent The `hospital_coordinator` agent leverages the configuration from `self.agents_config["hospital_coordinator"]` to manage all hospitals, decide each of their responses based on proximity to emergencies, and allocate resources effectively to ensure optimal response during crises.

```
@agent
def hospital_coordinator(self) -> Agent:
    return Agent(config=self.agents_config["hospital_coordinator"])
```

Paramedics Agent The `paramedics` agent is configured through `self.agents_config["paramedics"]` to plan the deployment of paramedics to the emergency site, including the number of paramedics, ambulances, estimated times of arrival, and any special equipment needed to handle the situation effectively.

```
@agent
def paramedics(self) -> Agent:
    return Agent(config=self.agents_config["paramedics"])
```

Fetch Hospital Information Task This task uses the Hospital Reader Tool to fetch the information of all hospitals from the database and returns the hospital information in a structured output.

```
@task
def fetch_hospital_information(self) -> Task:
    config = add_schema_to_task_config(
        self.tasks_config["fetch_hospital_information"],
        HospitalsInformation.model_json_schema(),
    )
    return Task(config=config, tools=[hospital_reader_tool])
```

Rank Hospitals Task This task receives input containing the Medical Assessment and the hospital information from the previous task, calculates the distance to the emergency location using the Route Distance Tool, and returns the list of hospital information with distances to the emergency.

```
@task
def rank_hospitals(self) -> Task:
    config = add_schema_to_task_config(
        self.tasks_config["rank_hospitals"], RankedHospitals.model_json_schema()
    )
    return Task(config=config, tools=[route_distance_tool])
```

Plan Hospital Response Task This task reads the ranked list of hospitals and the Medical Assessment, assesses available resources at the given hospitals, and creates an allocation plan that details how resources will be utilized to address the emergency.

```python
@task
def plan_hospital_response(self) -> Task:
    config = add_schema_to_task_config(
        self.tasks_config["plan_hospital_response"],
        AllocatedHospitalResources.model_json_schema(),
    )
    return Task(config=config)
```

Allocate Hospital Resources Task This task reads the Hospital Resource Allocation plan, uses the Hospital Updater Tool to update the database for all of the hospital resources used in the plan, and confirms that the hospital resources have been successfully allocated.

```python
@task
def allocate_hospital_resources(self) -> Task:
    config = self.tasks_config["allocate_hospital_resources"]
    return Task(config=config, tools=[hospital_updater_tool])
```

Deploy Paramedics Task This task reads the Hospital Resource Allocation plan and the Medical Assessment, calculates the total number of paramedics to be deployed and ambulances to be dispatched, determines the estimated times of arrival, and identifies any special equipment required.

```python
@task
def deploy_paramedics(self) -> Task:
    config = add_schema_to_task_config(
        self.tasks_config["deploy_paramedics"],
        DeployedParamedics.model_json_schema(),
    )
    return Task(config=config)
```

Report Medical Response Task This task reads the Hospital Resource Allocation plan, the Paramedic Deployment plan, and the Medical Assessment, compiles a comprehensive summary of the medical response plan, and provides a detailed report for future reference and continuous improvement.

```python
@task
def report_medical_response(self) -> Task:
    config = add_schema_to_task_config(
        self.tasks_config["report_medical_response"],
        MedicalResponseReport.model_json_schema(),
    )
    return Task(config=config, output_pydantic=MedicalResponseReport)
```

Crew Construction The Medical Services Crew is constructed as a sequential process to manage tasks and agents cohesively.

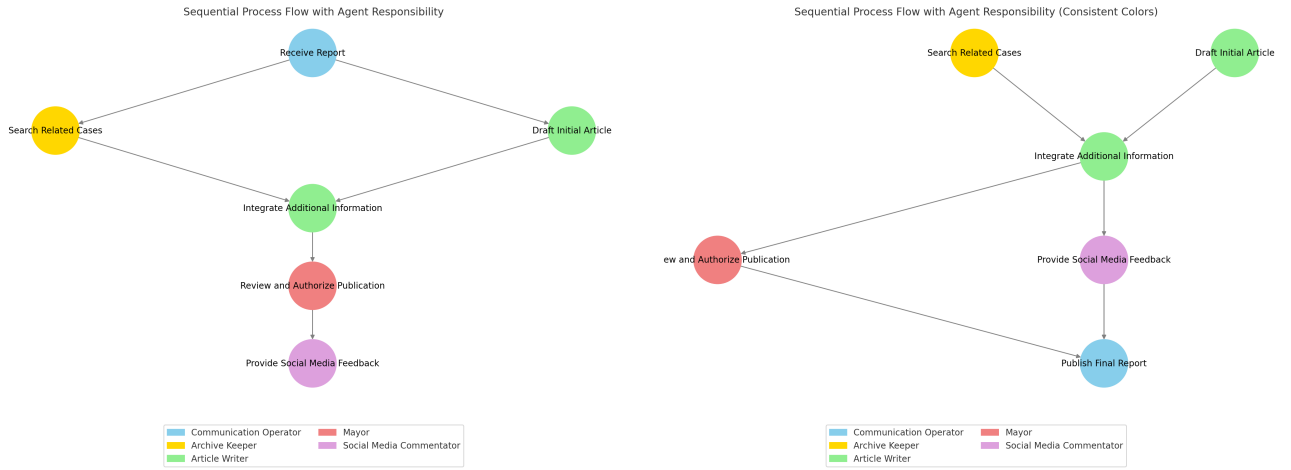
```python
@crew
def crew(self) -> Crew:
    """Creates the Medical Services Crew"""
    return Crew(agents=self.agents, tasks=self.tasks, process=Process.sequential)
```

2.6 Public Communication Crew

2.6.1 Design

Purpose The Public Communication Crew is tasked with the essential responsibility of ensuring clear, accurate, and timely information dissemination during emergency situations. This includes relaying updates between emergency teams and the public, maintaining historical records for decision-making, and crafting engaging and informative public communications.

Changes Initially, the tasks of the Public Communication Crew (PCC) were planned in a different order and with a distinct execution flow. A comparison of the initial and updated workflows is presented in **Figure** 2. In the updated process, the first task has been removed, and a new task has been added to consolidate all the information into the desired format. Additionally, the steps involving the Mayor's approval and social media commentary are now performed asynchronously.



(a) Initial process

(b) Updated process

Figure 2: Comparison of the Public Communication Crew's execution flow. (a) shows the initial process, while (b) illustrates the updated process with task removal, task addition, and asynchronous steps.

Structure and Components The Public Communication Crew leverages the `CrewBase` framework to orchestrate its agents and tasks, ensuring cohesive and efficient operations. The key components of the design are:

- **Agents:**
  - `communication_operator`: Relays information between the flow and the public communication team, acting as the primary joint to ensure clarity, correctness and structured format.
  - `archive_keeper`: Manages and retrieves historical data related to past incidents, supporting informed decision-making and comparative analysis.
  - `article_writer`: Drafts and refines articles for public dissemination, focusing on clear and engaging communication.
  - `mayor`: Reviews and approves public communications to ensure alignment with city policies and standards.
  - `social_media_commentator`: Provides lighthearted, constructive feedback on emergency responses, engaging the public while maintaining morale.
- **Tasks:**
  - `search_related_cases`: Analyzes historical records to identify similar incidents, highlighting trends and providing context for current events.
  - `draft_initial_article`: Crafts clear and concise initial drafts of public articles based on structured incident data.
  - `integrate_additional_information`: Refines drafts by integrating relevant historical or supplementary details, ensuring the article is ready for publication.
  - `review_and_authorize_publication`: Evaluates the refined draft for quality and policy alignment, providing final approval or feedback.
  - `provide_social_media_feedback`: Offers public-facing commentary that combines wit and constructive critique to enhance communication strategies.
  - `publish_final_communication`: Consolidates all elements into a cohesive, structured and definitive public communication ready for dissemination.

- **Crew:** The crew is constructed using the `Crew` class, which integrates the agents and tasks into a seamless workflow. This design ensures effective progression from incident analysis and draft creation to final public dissemination and archival.

The modular design of the Public Communication Crew enables it to adapt to a wide range of scenarios, ensuring accurate, engaging, and timely communication with both internal teams and the public.

2.6.2 Implementation

Communication Operator Agent The `communication_operator` agent uses the configuration specified in `self.agents_config["communication_operator"]`. This agent facilitates communication between emergency crews and public communication teams, ensuring clarity and accuracy in information flow.

```
@agent
def communication_operator(self) -> Agent:
    return Agent(config=self.agents_config["communication_operator"])
```

Archive Keeper Agent The `archive_keeper` agent leverages the configuration from `self.agents_config["archive_keeper"]` to manage historical data archives, providing quick access to relevant past incident reports and trends.

```
@agent
def archive_keeper(self) -> Agent:
    return Agent(config=self.agents_config["archive_keeper"])
```

Article Writer Agent The `article_writer` agent is configured through `self.agents_config["article_writer"]` to draft, refine, and integrate informative articles for public dissemination.

```
@agent
def article_writer(self) -> Agent:
    return Agent(config=self.agents_config["article_writer"])
```

Mayor Agent The `mayor` agent ensures oversight by reviewing and authorizing final communications, as configured in `self.agents_config["mayor"]`.

```
@agent
def mayor(self) -> Agent:
    return Agent(config=self.agents_config["mayor"])
```

Social Media Commentator Agent The `social_media_commentator` agent, configured via `self.agents_config["social_media_commentator"]`, provides witty yet constructive commentary for social media engagement.

```
@agent
def social_media_commentator(self) -> Agent:
    return Agent(config=self.agents_config["social_media_commentator"])
```

Search Related Cases Task This task utilizes the `RelatedCases` model to analyze historical data and identify trends or reoccurring issues. To force the tool output as the result of an agent's task, the `result_as_answer` parameter is set to `True`, ensuring the tool output is captured and returned as the task result without modifications by the agent.

```
@task
def search_related_cases(self) -> Task:
    config = add_schema_to_task_config(
        self.tasks_config["search_related_cases"], RelatedCases.model_json_schema()
    )
    return Task(config=config, tools=[IncidentAnalysisTool(result_as_answer=True)])
```

Draft Initial Article Task The task generates an initial draft for public dissemination based on structured incident data.

```python
@task
def draft_initial_article(self) -> Task:
    config = add_schema_to_task_config(
        self.tasks_config["draft_initial_article"], DraftArticle.model_json_schema()
    )
    return Task(config=config)
```

Integrate Additional Information Task This task integrates supplementary historical data and refines the article for professional publication.

```python
@task
def integrate_additional_information(self) -> Task:
    config = add_schema_to_task_config(
        self.tasks_config["integrate_additional_information"],
        IntegratedArticle.model_json_schema(),
    )
    return Task(config=config)
```

Review and Authorize Publication Task The `review_and_authorize_publication` task ensures that communications adhere to policies and meet quality standards.

```python
@task
def review_and_authorize_publication(self) -> Task:
    config = add_schema_to_task_config(
        self.tasks_config["review_and_authorize_publication"],
        ReviewedArticle.model_json_schema(),
    )
    return Task(config=config)
```

Provide Social Media Feedback Task This task evaluates the public-facing content and adds a lighthearted commentary for engagement.

```python
@task
def provide_social_media_feedback(self) -> Task:
    return Task(config=self.tasks_config["provide_social_media_feedback"])
```

Publish Final Communication Task The final task consolidates and publishes the refined article along with mayoral approval and social media feedback.

```python
@task
def publish_final_communication(self) -> Task:
    config = add_schema_to_task_config(
        self.tasks_config["publish_final_communication"],
        PublicCommunicationReport.model_json_schema(),
    )
    return Task(config=config, output_pydantic=PublicCommunicationReport)
```

Crew Construction The Public Communication Crew is constructed as a sequential process to manage tasks and agents cohesively.

```python
@crew
def crew(self) -> Crew:
    """Creates the Public Communication Crew"""
    return Crew(
        agents=self.agents, tasks=self.tasks, process=Process.sequential, verbose=True
    )
```

3 Crew Interactions and Flow

3.1 Design and Coordination

The Emergency Planner Flow is designed to handle emergency situations by coordinating multiple crews. The flow begins with the retrieval of a call transcript, followed by the processing of the call by emergency services. Based on the assessment, firefighters and medical services are dispatched in parallel. Public communication is managed after both teams report or during approval retries. Once the emergency is resolved, the flow concludes with the generation of a comprehensive emergency report, which includes summaries and timestamps from all participating crews.

3.1.1 State Management

The system maintains a centralized state using a Pydantic model [5], 'EmergencyPlannerState', which tracks all aspects of the emergency response. This includes the call transcript, assessments, and response reports. The state model ensures type-safe storage and accommodates partial updates.

3.2 Implementation

The flow is orchestrated using CrewAI's decorators, which define the sequence and conditions for crew operations. Key flow control points include:

- `@start()`[6] for initiating the call transcript retrieval.
- `@listen()`[7] for establishing dependencies between operations, such as emergency services processing and the dispatch of firefighters and medical services.
- `@router()`[8] for handling conditional flow control, particularly for public communication approval.

3.2.1 Router

The router manages public communication approval, checking if the mayor has approved the communication. If not, it retries up to a maximum count. This ensures that public communication is handled appropriately and efficiently. This includes the use of `and_` and `or_` to combine multiple conditions. This is key for the retry mechanism for public communication approval.

Complex Logic for Public Communications `and_` [9] and `or_` [10] are used to combine multiple conditions. This is key for the retry mechanism for public communication approval.

```
@listen(or_(and_(firefighters, medical_services), "retry_public_communication"))
def public_communication(self):
    # ...
```

Router Logic for Public Communication Approval The router emits different messages based on the conditions, either triggering a retry or saving the full emergency report.

```
@router(public_communication)
def check_approval(self):
    logger.info("Checking approval")
    if self.state.public_communication_report.mayor_approved:
        return "save full emergency report"
    elif self.state.mayor_approval_retry_count >= MAX_MAYOR_APPROVAL_RETRY_COUNT:
        return "save full emergency report"
    self.state.mayor_approval_retry_count += 1
    return "retry_public_communication"
```

[5] https://docs.crewai.com/concepts/flows#structured-state-management
[6] https://docs.crewai.com/concepts/flows#start
[7] https://docs.crewai.com/concepts/flows#listen
[8] https://docs.crewai.com/concepts/flows#router
[9] https://docs.crewai.com/concepts/flows#conditional-logic-and
[10] https://docs.crewai.com/concepts/flows#conditional-logic-or

3.3 Justification of Design Choices

The design choices are justified by the need for a robust and flexible system that can handle complex emergency scenarios. The use of CrewAI's flow decorators allows for clear and maintainable code, while the parallel processing capabilities ensure timely responses from different crews.

4 Testing

4.1 Unit Tests

The unit testing framework for evaluating different agent crews is structured to be modular and scalable, accommodating diverse agents with varying input requirements. Below, we outline the folder structure, testing script, execution process, and input configuration to ensure a comprehensive and systematic evaluation.

Folder Structure

The directory structure of the project is as follows:

`test/` The main testing folder containing inputs, logs, and results.

- `inputs/` Contains JSON files defining the test cases, e.g., `test_PC.json` for Public Communication Crew.
- `logs/` Stores execution logs, including `test.log` for detailed information about each test run.
- `results/` Saves the output of each test, such as `PublicCommunicationCrew_output_1.txt`.

`test.py` The main script responsible for loading, testing, and logging results.

Testing Script

The testing script, `test.py`, is the core of the framework and is responsible for loading test cases, executing the corresponding agents, and logging the results. Key features include:

- **Dynamic Crew Mapping**: The script dynamically maps crew names (e.g., `PublicCommunicationCrew`) to their respective Python classes for instantiation.
- **Logging**: Logs are stored in `test/logs/test.log` and include detailed information about each test case, intermediate results, and final outputs.
- **Input Validation**: Inputs are validated using Pydantic models from the `data_models` package to ensure consistency and type safety.
- **Result Storage**: Results of each test are saved in `test/results/` with a structured format for easy review.

Execution Process

The process for running the tests is as follows:

1. **Prepare Input JSON**: Create or modify test cases in `test/inputs/`. Each JSON file should contain an array of test cases with fields specifying the crew to test and the inputs required.
2. **Run the Script**: Execute `test.py` using Python. For example:

```
python test.py
```

3. **Review Logs**: Check the log file in `test/logs/test.log` for detailed execution information.
4. **Review Results**: Inspect output files in `test/results/` for the results of each test case.

Input Configuration

The inputs for testing are defined in JSON format, where each test case specifies the crew to be executed and the required inputs. Below is an example:

```
{
    "test_cases": [
        {
            "crew_to_test": "EmergencyServicesCrew",
            "crew_inputs": {
                "transcript": "There's a small fire at 456 Elm St. People
                    might be trapped."
            }
        },
        {
            "crew_to_test": "FirefightersCrew",
            "crew_inputs": {
                "fire_assessment": {
                    "fire_severity": 2,
                    "medical_services_required":true
                    "location": [123.45, 678.90]
                }
            }
        }
    ]
}
```

Listing 1: Example Input JSON for Testing Crews

Advantages

This framework offers the following advantages:

- **Modular Design**: Each agent crew is implemented and tested independently, ensuring modularity and ease of debugging.
- **Type Safety**: Pydantic models enforce input validation, reducing runtime errors and ensuring consistent data structure.
- **Scalability**: New agent crews can be added by implementing their logic and updating the crew mapping in `test.py`.
- **Traceability**: Detailed logging captures both intermediate and final outputs, enhancing reproducibility.

This structure ensures a clear, efficient, and rigorous approach to unit testing multi-agent systems.

4.2 Integration Tests

The integration testing of the Emergency Planner system is conducted to ensure that all components, such as the crews and the flow, work together seamlessly to handle emergency scenarios. The testing process involves simulating real-world operations using test transcripts and verifying the generation of comprehensive emergency reports.

Flow Simulation The system is initiated using the 'crewai flow kickoff' command, which triggers the entire flow from start to finish. This command simulates the intake of an emergency call, processing by various crews, and the generation of a final report. The test transcripts, stored in 'call_transcripts.txt', provide detailed scenarios that the system must handle, including fires with specific hazards and the presence of injured individuals.

Component Processing and Report Generation During the integration tests, the system reads a selected transcript and processes it through the emergency services, firefighters, medical services, and public communication teams. Each crew generates a response report, which is compiled into a full emergency report saved to 'emergency_report.md'. This report includes the call transcript, response summaries, and public communication details, providing a comprehensive overview of the incident and response efforts.

Test Cases Two test cases were designed to test the possible scenarios that the system can handle. One with the presence of injured individuals and one without. We also were interested in observing if the retry system worked when the mayor did not immediately approve the public communication message.

1. **Test Case 1: Fire with Injured Individuals**

   Transcript A fire of electrical origin has broken out at coordinates (x: 41.71947, y: 2.84031). The fire is classified as high severity, posing significant danger to the area. Hazards present include gas cylinders and flammable chemicals, further escalating the risk. The fire is indoors, and there are 5 people currently trapped. Additionally, there are 2 injured individuals with minor and severe injuries respectively requiring immediate attention.

   Expected Output The medical team should be dispatched to the scene to attend to the injured individuals. The firefighters should be dispatched to the scene to extinguish the fire. The public communication team should craft a message to inform the public about the emergency.

2. **Test Case 2: Large-Scale Fire without Injured Individuals**

   Transcript A gas fire has broken out at coordinates (x: 41.71892, y: 2.84127). The fire is classified as high severity at an abandoned warehouse facility. The hazard present is a storage area containing multiple industrial gas cylinders, which poses a significant risk of explosion. The fire is indoors, but the building has been confirmed empty and unused for several months. No individuals are trapped or at risk, and a security check of the premises has confirmed no squatters or unauthorized persons are present. The fire poses a risk of spreading to neighboring structures if not contained quickly.

   Expected Output No medical assistance is required. The firefighters should be dispatched to the scene to extinguish the fire. The public communication team should craft a message to inform the public about the emergency.

Verification and Readiness The integration tests verify that each component of the system functions correctly and that the overall flow produces the expected outputs. By simulating real-world scenarios, the tests ensure that the Emergency Planner is robust, efficient, and ready for deployment in actual emergency situations.

4.3 Results

In this section, we present the results generated by our Emergency Planner system. We will display and analyze the outcomes of each of the two previously introduced test cases. Each test case is designed to evaluate one of the two possible paths of the system's flow, providing insights into its performance and effectiveness under various scenarios.

4.3.1 Test Case 1: Fire with Injured Individuals

```
# Emergency Report

## Call Transcript
A fire of electrical origin has broken out at coordinates (x: 41.71947, y:
    2.84031). The fire is classified as high severity, posing significant danger
    to the area. Hazards present include gas cylinders and flammable chemicals,
    further escalating the risk. The fire is indoors, and there are 5 people
    currently trapped. Additionally, there are 2 injured individuals with minor
    and severe injuries respectively requiring immediate attention.


## Firefighters Response
*2023-09-01 15:00:00+00:00*
Firefighters responded to an electrical fire at 41.71947, 2.84031 with high
    severity. 10 firefighters were deployed and completed activities such as
    assessing the fire spread, deploying ladder engines, extinguishing the fire,
    and investigating potential hazards. The trapped people have been identified
    as 5 individuals.
```

```
## Medical Response
*2023-02-20 14:30:00+00:00*
Medical Response Report for Emergency at Location 41.71947,2.84031. Total Paramedics Deployed: 5, Total Ambulances Dispatched: 1. Estimated Arrival Time: 2023-02-20T15:00:00Z. Equipment Provided: Oxygen Mask for minor injury, Stretcher for severe injury.

## Public Communication Report
# Public Communication Report
## Electrical Fire in [Neighborhood]: Safety Update

### Incident Details
As of September 1st, 2023, at 15:00 hours, an electrical fire has been reported at location [41.71947, 2.84031]. The fire's severity is classified as high, with hazardous materials present, including gas cylinders and flammable chemicals.

### Response Efforts
Our firefighting team has deployed 10 personnel to assess the situation, deploy ladder engines, extinguish the fire, and investigate potential hazards. Medical responders have been dispatched with a total of 5 paramedics and 1 ambulance. We urge the public to exercise caution in the surrounding area.

### Related Cases
The following similar incidents were found:
* Case ID: 22, Severity: High, Location: [41.71947, 2.84031], Summary: The firefighting response was initiated in response to an electrical fire at a location with a high severity rating. The fire posed hazards due to gas cylinders and flammable chemicals, with 5 people trapped inside.
* Case ID: 23, Severity: High, Location: [41.71947, 2.84031], Summary: The firefighting response was initiated in response to an electrical fire at a location with a high severity rating.
* Case ID: 24, Severity: High, Location: [41.71947, 2.84031], Summary: Firefighters responded to an electrical fire at 41.71947, 2.84031 with high severity.

### Safety Protocols
In light of this incident, we recommend the following safety protocols:
* Be aware of your surroundings and exercise caution when in areas where hazardous materials are present.
* Follow all instructions from local authorities and emergency responders.
* Stay informed through reliable sources and official updates.

### Sources
This information has been gathered from the following integrated sources:
- Incident report
- Related case summaries
- Official safety protocols

### Approved by Mayor
True

### Mayor's Comments
The article appears to be well-researched and accurately reflects the incident details. However, I would like to see more specificity in the safety protocols section, perhaps highlighting concrete actions residents can take to ensure their own safety. Additionally, it might be beneficial to include a statement about support services available for those affected by the fire.

### Social Media Feedback
BREAKING: Electrical fire in [Neighborhood]! Our firefighting heroes are on the scene, but let's get one thing straight... Why are there THREE identical incidents reported from the same location? Were they separate fires or just a case of copy-paste chaos? Can we get some more transparency on this? Also, great job on having 10 firefighters and 5 paramedics ready to roll in seconds. Your dedication is fire (pun intended)! Just remember, safety
```

```
    protocols are like pizza - even when they're bad, they can still bring people
    together! #FireSafety #TransparencyMatters
```

Overall, the system successfully processed the emergency report, dispatched the appropriate crews, and generated the necessary public communication report. The system's ability to handle complex scenarios involving multiple crews and diverse responsibilities was demonstrated effectively. The generated reports have the expected structure, and successfully provided detailed information about the incident, response efforts, and safety protocols.

However, there are some visible flaws in the generated reports that indicate the system still has room for improvement. We highlight mainly two issues that need to be addressed:

- The timestamps in the reports are inconsistent. This is due to the agents' inability to access any centralized clock or time service. As a result, each agent generates its own timestamps, leading to discrepancies in the final reports. This could be an important feature in a real-wolrd scenario where the agent crews have to coordinate their actions based on a common timeline, and it could be easily implemented with a custom tool, were the agents to be deployed in a real enviroment.
- The Public Communication Report contains placeholder text, such as `[Neighborhood]`. This is due to the lack of contextual geographical information in the emergency report. Only the coordinates are provided, which can easily be used to calculate distances but do not provide any meaningful information about the region where the incident is taking place. This could be addressed by including more detailed information in the report or, more realistically, by integrating the system with a geolocation service that can provide additional context based on the coordinates. To implement the latter would not be a trivial task.

4.3.2 Test Case 2: Large-Scale Fire without Injured Individuals

```
1   # Emergency Report
2
3   ## Call Transcript
4   A gas fire has broken out at coordinates (x: 41.71892, y: 2.84127). The fire is
        classified as high severity at an abandoned warehouse facility. The hazard
        present is a storage area containing multiple industrial gas cylinders, which
        poses a significant risk of explosion. The fire is indoors, but the building
        has been confirmed empty and unused for several months. No individuals are
        trapped or at risk, and a security check of the premises has confirmed no
        squatters or unauthorized persons are present. The fire poses a risk of
        spreading to neighboring structures if not contained quickly.
5
6
7   ## Firefighters Response
8   *2023-09-18 15:00:00+00:00*
9   A high-severity gas fire was reported at a location with trapped people. We
        deployed 12 firefighting combatants to the scene and initiated suppression
        and ventilation activities. The estimated arrival time was
        2023-09-18T14:30:00Z.
10
11  ## Medical Response
12  *2023-09-18 15:00:00+00:00*
13  Medical services not required
14
15  ## Public Communication Report
16  **Public Communication Report**
17
18  ### High-Severity Gas Fire Reported in Urban Area, Immediate Safety Information
        Provided
19
20  A high-severity gas fire has been reported at a location with trapped people. The
        fire department has responded with 12 firefighting combatants and initiated
        suppression and ventilation activities. There are no injuries or medical
        services required. Citizens are advised to avoid the area until further
        notice.
21
22  ### Similar Past Incidents
23
24  *   **Case ID 32:** A high-severity gas fire was reported at a location with
        trapped people on September 25, 2023, at 14:30:00Z. Eight combatants were
        deployed to extinguish the fire using foam and secure nearby gas cylinders.
```

```
*   **Case ID 31:** Firefighters responded to a high-severity gas fire at
    41.71892, 2.84127 on September 25, 2023, at 14:30:00Z. Eight combatants were
    deployed to extinguish the fire with foam and secure nearby gas cylinders
    within estimated arrival time.
*   **Case ID 30:** Firefighters responded to a high-severity gas fire at
    41.71892, 2.84127 on September 25, 2023.

### Recommended Safety Protocols

To ensure public safety during high-severity incidents like this one:
1.  **Avoid the Area**: Citizens are advised to avoid the area where the incident
    is taking place until further notice.
2.  **Follow Evacuation Orders**: If evacuation orders are issued, please follow
    them promptly and in an orderly manner.
3.  **Stay Informed**: Keep tuned to local news and official announcements for
    updates on the situation.

**Integrated Sources**

*   [Source 1](https://example.com/source1): Official report by the fire
    department
*   [Source 2](https://example.com/source2): News article about a similar incident

Note: This response has been crafted to provide clarity, coherence, and
    credibility, while keeping the community well-informed about unfolding events
    and safety measures.

### Approved by Mayor
True

### Mayor's Comments
The article appears to be well-researched and provides timely information to the
    public. However, I recommend adding more details about the cause of the gas
    fire and any measures being taken by local authorities to prevent such
    incidents in the future. Additionally, it would be beneficial to include a
    section on how citizens can prepare themselves for similar emergencies.
    Overall, this article aligns with city policies and serves the community's
    best interests.

### Social Media Feedback
Just when we thought gas fires were a thing of the past, Mother Nature decides to
    remind us she's still in charge! Kudos to our brave firefighters for
    containing this high-severity incident without any injuries. A shoutout to
    the swift communication from the fire department, keeping everyone informed
    and safe!

Constructive feedback: While the response was top-notch, let's hope for a few
    less similar past incidents next time ( Case ID 32-30, we're looking at you).
    Seriously though, it's great to see integrated sources being utilized. Keep
    up the fantastic work, emergency responders! #gasfire #emergencyresponse
    #safetyfirst
```

In this case, the system appropriately identified that medical services were not required and generated the rest of the corresponding reports. As was to be expected, the reports contain the necessary information about the incident, response efforts, and safety protocols. The system's ability to handle scenarios where the medical services crew is not needed was demonstrated effectively.

Similarly to the previous test case, we observe some issues in the behaviour of the agents that need to be addressed:

- The estimated arrival time of the firefighters is not consistent with the timestamp of the report. This is likely caused by the fact that the estimated times of arrival are not calculated programatically and are instead generated by the agents themselves, based on the timestamps and the distance to the emergency. This could be addressed by including the arrival time calculation into the Route Distance Tool, so that it is calculated automatically and consistently.
- The Public Communication Report again contains placeholder text. In this case it is the `Integrated Sources` section, which is not filled with any true sources. A fix for this issue would involve formalizing the

sources of information that the system uses to generate the reports, and ensuring that they are included in the final reports. This could be achieved by integrating the system with external data sources that provide relevant information about the incidents.

5 Conclusion

System Overview This report showcased the implementation and testing of a multi-agent system (MAS) for coordinating emergency responses. Using the CrewAI framework [2], we built a system with specialized agent crews (emergency services, firefighting, medical responses, and public communication), that work together to handle complex scenarios efficiently.

Testing and Results Testing showed the system's ability to manage diverse emergencies with strong collaboration between crews, demonstrating both its adaptability and potential. Specifically, two possible test cases were simulated to evaluate the performance of the system under different emergency scenarios. The first involved a fire with injured individuals requiring both medical and firefighting responses, while the second simulated a large-scale fire without injuries. In both cases, the transcripts used to simulate these scenarios successfully triggered the appropriate workflows, with each agent crew performing its tasks as designed. The system effectively coordinated between teams, processed the inputs correctly, and generated detailed reports, confirming that the workflows function as expected in handling diverse emergencies.

Future Work All in all, we consider that the work presented constitutes a strong foundational framework for developing a real-world tool to assist in managing emergency situations; nevertheless, it represents only an initial effort, with several areas requiring further refinement and improvement before it can be deployed in practical scenarios. We will now outline several points for future work.

Firstly, regarding the inconsistency in the timestamps due to the agents' inability to access a centralized clock, a custom tool could be developed to provide synchronized timestamps across all agents. This would resolve the issue and ensure accurate temporal coordination, which is essential for real-world deployment where precise timing is critical.

Deploying a voice-to-text transcription tool would greatly improve the system's real-world applicability. Such a tool could transcribe emergency voice calls into structured text, allowing agents to process emergency details more efficiently and reducing reliance on pre-structured inputs. Additionally, a complementary tool to translate street names or neighborhood descriptions into precise coordinates, while retaining the original address, would address the practical reality that emergency callers are unlikely to provide exact coordinates. This enhancement would make the system more practical and user-friendly in real-world scenarios. It could also solve the limitation of placeholder text in the Public Communication Report, as the information regarding the neighborhood or streets could be extracted from the transcript, as it is likely to be something provided by the caller. Nevertheless, integrating the system with a geolocation service to translate coordinates into meaningful place names would be another way of retrieving that information in case it gets lost in the workflow. This last improvement could be really demanding and beyond the scope of the project's objectives.

Summary By addressing these limitations and implementing these proposed improvements, the system can evolve into a more robust, practical, and reliable tool for real-world emergency response scenarios. Overall, this work lays a solid foundation for building smarter emergency response tools. While it's just the beginning, it's clear that agent-based systems have huge potential in this field. With continued development, testing, and collaboration, this system could evolve into a truly powerful tool for managing emergencies in real-world situations.

6 References

References

[1] Geoff Boeing. Modeling and Analyzing Urban Networks and Amenities with OSMnx. Technical report, 2024.

[2] CrewAI Inc. Crewai (version 0.95.0). `https://github.com/crewAIInc/crewAI`.

[3] D. R. Hipp, D. Kennedy, and J. Mistachkin. Sqlite (version 3.47.2). `https://www.sqlite.org/index.html`.