Emergency Response: Cooperation and Coordination Mechanisms in Multi-Agent Systems

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

December 7, 2024

Contents

1 Introduction

In this report, we propose a robust framework to address the emergency response problem using cooperation and coordination mechanisms in multi-agent systems (MAS). The system, dubbed CrewAI, orchestrates diverse specialized crews to handle emergencies through structured processes, ensuring efficiency, precision, and adaptability. The approach integrates process definition, pydantic outputs, and agent interaction, enabling smooth task execution and seamless inter-crew communication.

The **process definition** outlines the workflows for individual crews, specifying sequential and parallel dependencies, agent roles, and key operational details. Each crew is equipped with tailored processes to ensure timely and effective responses.

Pydantic outputs ensure consistency and clarity by defining structured data models for task outputs. These models are pivotal in facilitating inter-crew interactions and ensuring data integrity throughout the emergency response lifecycle.

Finally, the **agent interaction** mechanism employs flow-based coordination, leveraging CrewAI's advanced routing and state management capabilities. Through logical operators and conditional triggers, the framework ensures synchronization across crews, optimizing both task allocation and response accuracy.

This comprehensive report details each component of the framework, emphasizing its adaptability and potential scalability. It serves as a blueprint for integrating multi-agent coordination into real-world emergency management systems.

2 Process Definition

2.1 Emergency Services Crew

1. **Receive and Assess Call.** The *Emergency Call Agent* receives incoming calls and collects relevant details about the incident. The information that this agent receives answers the following six questions and is saved in a report:

    - What type of fire is it? E.g. ordinary, electrical, gas, etc.
    - Where is it? The location is received as coordinates (x, y).
    - Is anyone injured? How badly? The answer will be a list of strings, detailing the risk level of each person. If the list if empty then there will be no injured people and it will be unnecessary to report it to the *Medical Service Crew.*
    - How severe is the fire? It will be considered as low, medium or high.
    - Are there hazards? Examples of hazards could include gas cylinders, chemicals, explosions, etc.
    - Is it an indoor or outdoor fire? The answer will be either *outdoor* or *indoor.*
    - Is anyone inside or trapped? The answer will be an integer number M representing the number of trapped people. If $M > 0$, rescues are needed, and the *Notification Agent* will detail that to the Fire Fighters Crew.

2. **Notify Other Crews Decision.** The *Notification Agent* receives the details about the fire then it decides which crew should be notify and send all the information to the flow:

    - Information provided to the Flow:
        - Fire type.
        - Location (x, y).
        - Details of injured individuals.
        - Fire severity.
        - Hazards.
        - If the fire is indoors or outdoors.
        - Number of trapped people M.
        - If medical services are required, this condition will only be true if the list of *details of injured individuals* is not empty.

Task Dependencies: The sequential workflow for the Emergency Services Crew depends on task dependencies to ensure efficiency and coordination:

- The *Notify Other Crews Task* depends on the completion of the *Receive and Assess Call Task*.

The task dependencies and agents who perform each task can be observed in Figure 1.

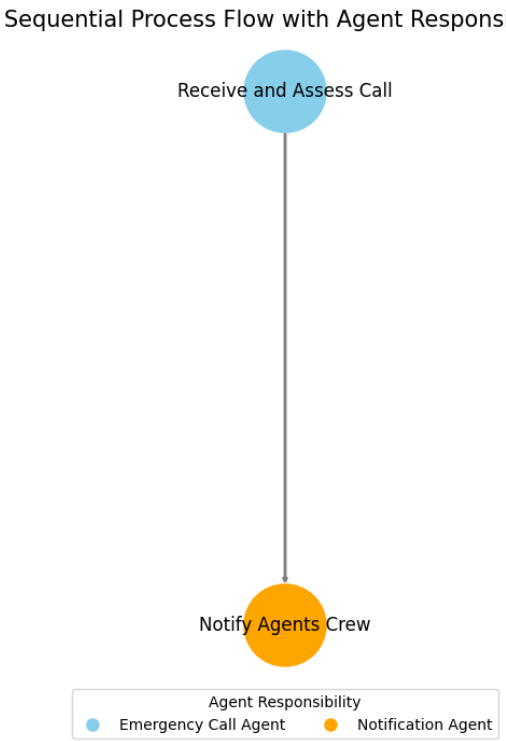

Figure 1: Sequential Process Flow of the Medical Services Crew with Agent Responsibilities

2.2 Firefighter Agent Crew

The Firefighter Agent Crew operates within a structured **sequential process** to ensure effective and coordinated response to fire emergencies. Each task is assigned to a specific agent with well-defined responsibilities, as detailed below:

1. **Receive Report:** The *Fire Chief* receives a fire report from the Emergency Service Operator. This serves as the starting point of the process, containing critical information such as the location and severity of the fire.
2. **Allocate Firefighting Resources:** The *Equipment Technician* determines if there exact resources required to combat the fire in question.
3. **Deploy Fire Combatants:** The *Fire Combatants* are deployed to the place of the fire, reporting an estimation of the time of arrival and a list of the fire fighting activities that will have to be performed.
4. **Report Firefighting Response:** The *Fire Chief* reports back a comprehensive summary of the firefighting activities.

Task Dependencies The sequential process relies on strict task dependencies to maintain an organized workflow:

- *Allocate Firefighting Resources* depends on the completion of *Receive Report*.
- *Deploy Fire Combatants* depends on the completion of *Deploy Fire Combatants*.
- *Report Firefighting Response* depends on the completion of *Deploy Fire Combatants*.

The visual representation in Figure 2 highlights these dependencies and assigns colors to denote the responsible agents.

2.3 Medical Services Crew

The Medical Services Crew operates follows a **sequential** task structure to plan the treatment and evacuation of injured people from the emergency site. The tasks included within the Medical Services are:

1. **Receive Report:** The *Medical Services Operator* receives the medical report of the fire incident, and parses key information, such as the location, the number of injured, and the severity of injuries.
2. **Rank Hospitals:** The *Hospital Coordinator* ranks the city's hospitals based on distance to the emergency location.
3. **Allocate Hospital Resources:** The *Hospital Coordinator* assesses the available resources (beds, ambulances, paramedics) at the hospitals, and allocates their resources according to the needs of the emergency.
4. **Deploy Paramedics:** The *Paramedics* plan their deployment to the place of the incident, reporting the total number of paramedics and ambulances dispatched, as well as their estimated times of arrival, and any special equipment that they could need.
5. **Report Medical Response:** The *Medical Services Operator* reports back a comprehensive summary of the response plan.

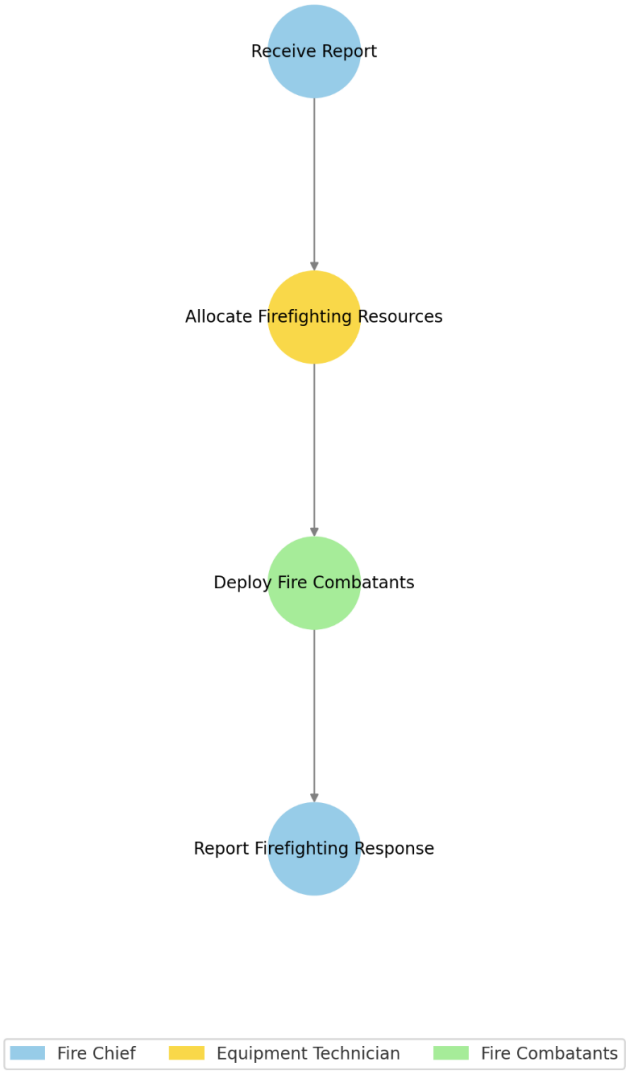

Figure 2: Sequential Process Flow of the Firefighter Crew with Agent Responsibilities

Task Dependencies The sequential nature of the process requires to establish task dependencies to define the crew's workflow:

- The *Rank Hospitals* task depends on the completion of the *Recieve Report* task.
- The *Allocate Hospital Resources* task depends on the completion of *Rank Hospitals*.
- The *Deploy Paramedics* task depends on the completion of *Allocate Hospital Resources*.
- The *Report Medical Response* task depends on the completion of *Deploy Paramedics*.

The task dependencies and agents who perform each task can be observed in Figure 3.

2.4 Public Communication Crew

The Public Communication Crew operates within a structured **sequential process** to ensure efficient and accurate communication of fire incident reports to the public. Each task is assigned to a specific agent with well-defined responsibilities, as detailed below:

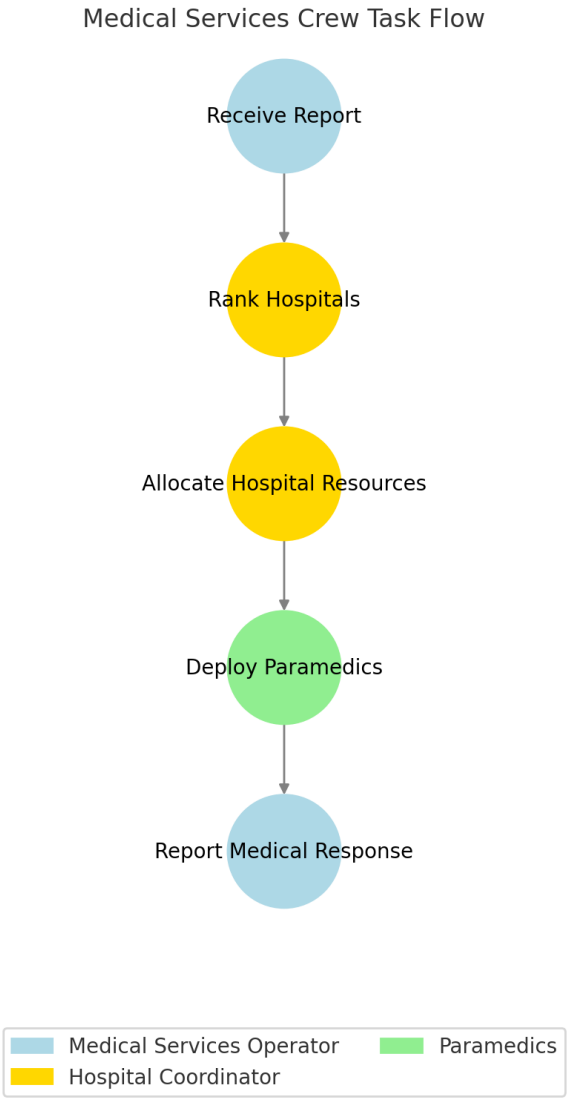

Figure 3: Sequential Process Flow of the Medical Services Crew with Agent Responsibilities

1. **Receive Report:** The *Communication Operator* obtains the fire incident report in Markdown format. This serves as the starting point for the process and can filter any information that is not relevant for this crew.

2. **Search Related Cases:** The *Archive Keeper* searches for past incidents with similar locations or fire types. This task depends on the completion of the *Receive Report* task.

3. **Draft Initial Article:** The *Article Writer* drafts an initial article based on the current report. This task also depends on the completion of the *Receive Report* task.

4. **Integrate Additional Information:** The *Article Writer* integrates insights from related cases into the draft. This task requires the completion of both the *Search Related Cases* and *Draft Initial Article* tasks.

5. **Review and Authorize Publication:** The *Mayor* reviews the article and either authorizes publication or provides feedback for revisions. This task depends on the completion of the *Integrate Additional Information* task.

6. **Provide Social Media Feedback:** The *Social Media Commentator* critiques the emergency response in a humorous yet constructive manner. This task depends on the approval of the article by the *Mayor*.

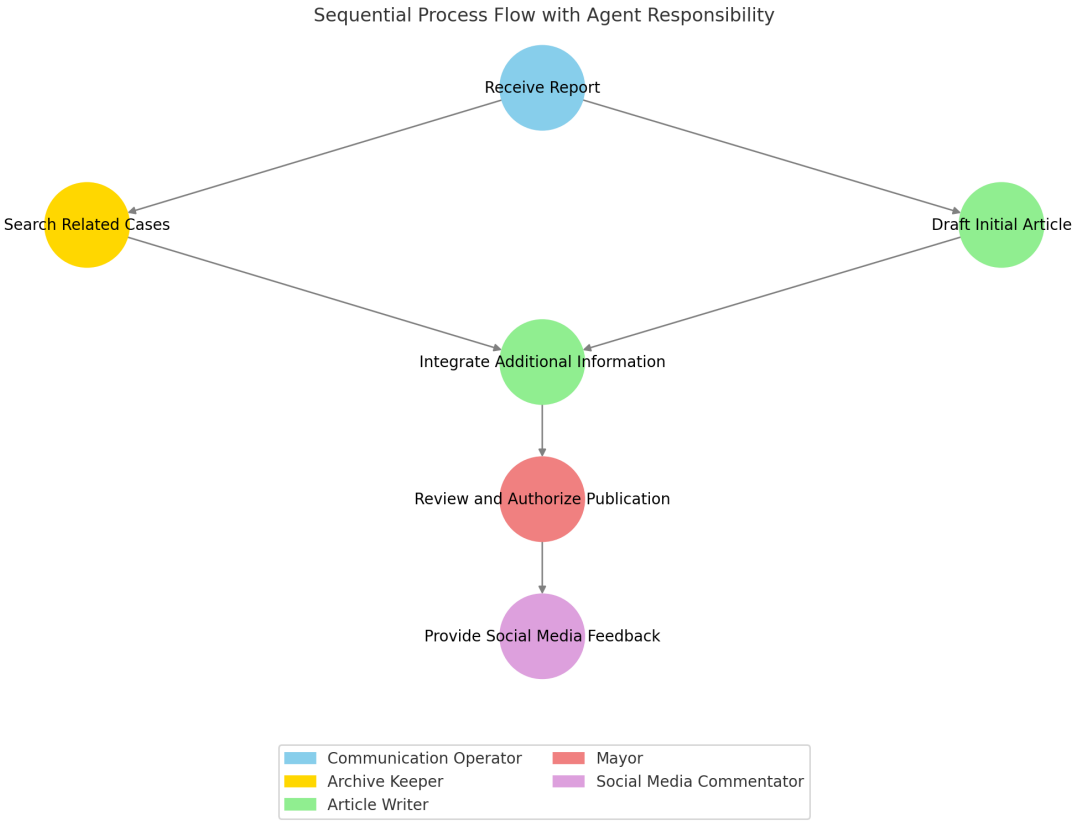

Figure 4: Sequential Process Flow of the Public Communication Crew with Agent Responsibilities

Task Dependencies The sequential process relies on strict task dependencies to ensure an organized workflow:

- *Search Related Cases* and *Draft Initial Article* can be executed in parallel but both depend on *Receive Report*.
- *Integrate Additional Information* requires the completion of both *Search Related Cases* and *Draft Initial Article*.
- *Review and Authorize Publication* depends on *Integrate Additional Information*.
- *Provide Social Media Feedback* requires article approval from the *Mayor*.

The visual representation in Figure 4 highlights these dependencies and assigns colors to denote the responsible agents, ensuring clarity and accountability.

3 Pydantic Outputs

Structured outputs are essential for ensuring clarity and consistency in task execution. Below are listed the Pydantic models used in the system.

3.1 Emergency Services Crew

Structured outputs ensure accurate information handling and effective communication within the Emergency Services Crew. Below are the Pydantic models designed for each task's output.

3.1.1 Receive and Assess Call Task Output

```python
class EmergencyDetails(BaseModel):
    fire_type: FireType  # Type of fire (e.g., ordinary, electrical, gas, etc.)
    location: Location  # Coordinates (x, y)
    injured_details: List[InjuryType]  # List of risk levels of injured people
    fire_severity: FireSeverity  # Severity of fire: low, medium, or high
    hazards: List[HazardType]  # Hazards present, e.g., gas cylinders, chemicals
    indoor: bool  # True if fire is indoor, False otherwise
    trapped_people: int  # Number of people trapped (0 if none)
```

Listing 1: Pydantic model for Receive and Assess Call Task Output

3.1.2 Notify Other Crews Task Output

```python
class CallAssessment(BaseModel):
    fire_type: FireType
    location: Location
    injured_details: List[InjuryType]
    fire_severity: FireSeverity
    hazards: List[HazardType]
    indoor: bool
    trapped_people: int
    medical_services_required: bool  # True if medical services are required, False otherwise
```

Listing 2: Pydantic model for Notify Other Crews Task Output

Summary of Outputs The Pydantic outputs for the *Emergency Services Crew* ensure structured data handling and effective communication between agents. Below is a summary of the outputs for each task:

- **Receive and Assess Call Task Output:** Captures critical incident details including fire type, location, injured details, severity, hazards, indoor/outdoor status, and trapped individuals.
- **Notify Other Crews Task Output:** Adds to the *Call Assessment* model to include information about whether medical services are required.

These structured models enhance precision and ensure clear communication throughout the emergency response processes.

3.2 Firefighter Agent Crew

Structured outputs ensure effective communication and accountability among team members in the Firefighter Agent Crew. Below are the Pydantic models designed to encapsulate outputs for each task in the firefighting process:

3.2.1 Receive Report Task Output

```python
class FireAssessment(BaseModel):
    location: Location  # Coordinates (x, y)
    fire_type: FireType  # Type of fire fire_severity
    fire_severity: FireSeverity  # Severity of fire: low, medium,
       or high
    trapped_people: int  # Number of trapped individuals
    hazards: List[HazardType]  # Hazards present
    hazards_present_indoor: bool  # True if fire is indoor, False
       otherwise
```

Listing 3: Pydantic model for Receive Report Task Output

3.2.2 Allocate Firefighting Resources Task Output

```python
class FireFightingMaterial(BaseModel):
    material_name: Literal[
        "pickup_truck",
        "ladder_engine",
        "water_tanker",
        "foam_tanker",
        "dry_chemical_tanker",
        "air_tanker",
    ]
    material_quantity: int

class AllocatedFirefightingResources(BaseModel):
    fire_assessment: FireAssessment
    resources: List[FireFightingMaterial]
```

Listing 4: Pydantic model for Allocate Firefighting Resources Task Output

3.2.3 Deploy Fire Combatants Task Output

```python
class FirefightingActivity(BaseModel):
    firefighting_activity: str
    priority: Literal["low", "medium", "high"]

class DeployedFireCombatants(BaseModel):
    fire_assessment: FireAssessment
    firecombatants_deployed: int
    estimated_arrival_time: datetime
    firefighting_activities: List[FirefightingActivity]
```

Listing 5: Pydantic model for Deploy Fire Combatants Task Output

3.2.4 Report Firefighting Response Task Output

```python
class FirefightersResponseReport(BaseModel):
    summary: str
    timestamp: datetime
```

Listing 6: Pydantic model for Report Firefighting Response Task Output

Summary of Outputs

- **Receive Fire Report Task Output:** Captures the essential details from the initial fire report, including fire type, severity, hazards, and any trapped individuals.
- **Allocate Firefighting Resources Task Output:** Documents the allocation of firefighting materials, including quantities and resource types.
- **Deploy Fire Combatants Task Output:** Tracks the deployment of personnel, estimated arrival times, and prioritized firefighting activities.
- **Report Firefighting Response Task Output:** Summarizes the firefighting response plan.

3.3 Medical Services Crew

Structured outputs ensure consistency and facilitate effective collaboration among agents within the Medical Services Crew. Below are the Pydantic models for each task's output:

3.3.1 Receive Report Task Output

```python
class MedicalAssessment(BaseModel):
    location: Location  # Coordinates (x, y)
    injured_details: List[InjuryType]  # List of risk levels of
        injured people
    fire_severity: FireSeverity  # Severity of fire: low, medium,
        or high
    hazards: List[HazardType]  # Hazards present, e.g., gas
        cylinders, chemicals
```

Listing 7: Pydantic model for Receive Report Task Output

3.3.2 Rank Hospitals Task Output

```python
class Hospital(BaseModel):
    hospital_id: str
    location: Location
    available_beds: int
```

```
    available_ambulances: int
    available_paramedics: int

class RankedHospitals(BaseModel):
    medical_assessment: MedicalAssessment
    ranked_hospitals: List[Hospital]
    timestamp: datetime
```

Listing 8: Pydantic model for Rank Hospitals Task Output

3.3.3 Allocate Hospital Resources Task Output

```
class HospitalResources(BaseModel):
    hospital_id: str
    beds_reserved: int
    ambulances_dispatched: int
    paramedics_deployed: int

class AllocatedHospitalResources(BaseModel):
    medical_assessment: MedicalAssessment
    hospital_resource_allocation: List[HospitalResources]
    timestamp: datetime
```

Listing 9: Pydantic model for Allocate Hospital Resources Task Output

3.3.4 Deploy Paramedics Task Output

```
class MedicalEquipment(BaseModel):
    equipment_name: Literal[
        "oxygen_mask",
        "stretcher",
        "defibrillator",
        "IV_drip",
        "other",
    ]
    use_case: str


class DeployedParamedics(BaseModel):
    medical_assessment: MedicalAssessment
    total_paramedics_deployed: int
    total_ambulances_dispatched: int
    estimated_arrival_times: List[datetime]
    equipment: List[MedicalEquipment]
```

Listing 10: Pydantic model for Deploy Paramedics Task Output

3.3.5 Report Medical Response Task Output

```python
class MedicalResponseReport(BaseModel):
    summary: str
    timestamp: datetime
```

Listing 11: Pydantic model for Report Medical Response Task Output

Summary of Outputs

- **Receive Report Task Output:** Captures the key details of the fire incident, including injury data.
- **Rank Hospitals Task Output:** Ranks the available hospital based on distance to the emergency site.
- **Allocate Hospital Resources Task Output:** Summarizes the resources provided by each hospital for emergency medical care.
- **Deploy Paramedics Task Output:** Reports the deployment plan, estimated times of arrival of each ambulance, and special medical equipment to be brought.
- **Report Medical Response Task Output:** Provides an overall response plan.

3.4 Public Communication Crew

Structured outputs are crucial for ensuring clarity, consistency, and seamless integration across tasks. Below are the Pydantic models designed for the tasks in the Public Communication Crew process:

3.4.1 Receive Report Task Output

```python
class EmergencyReport(BaseModel):
    call_assessment: CallAssessment
    firefighters_response_report: FirefightersResponseReport
    medical_response_report: MedicalResponseReport
    timestamp: datetime
    fire_severity: FireSeverity
    location_x: float
    location_y: float
```

Listing 12: Pydantic model for Receive Report Task Output

3.4.2 Search Related Cases Task Output

```python
class RelatedCase(BaseModel):
    case_id: int
    fire_severity: FireSeverity
    location_x: float
    location_y: float
    summary: str

```

```python

class RelatedCases(BaseModel):
    related_cases: List[RelatedCase]
```

Listing 13: Pydantic model for Search Related Cases Task Output

3.4.3 Draft Initial Article Task Output

```python
class DraftArticle(BaseModel):
    title: str
    public_communication_report: str
```

Listing 14: Pydantic model for Draft Initial Article Task Output

3.4.4 Integrate Additional Information Task Output

```python
class IntegratedArticle(BaseModel):
    public_communication_report: str
    integrated_sources: List[str]
```

Listing 15: Pydantic model for Integrate Additional Information Task Output

3.4.5 Review and Authorize Publication Task Output

```python
class ReviewedArticle(BaseModel):
    public_communication_report: str
    mayor_approved: bool
    mayor_comments: str
```

Listing 16: Pydantic model for Review and Authorize Publication Task Output

3.4.6 Provide Social Media Feedback Task Output

```python
class PublicCommunicationReport(BaseModel):
    public_communication_report: str
    mayor_approved: bool
    mayor_comments: str
    social_media_feedback: str
```

Listing 17: Pydantic model for Provide Social Media Feedback Task Output

Summary of Outputs

- **Receive Report Task Output:** Captures the initial fire incident report relevant details from *Emergency Services Crew*, *Firefighters Crew*, and *Medical Services Crew*.
- **Search Related Cases Task Output:** Retrieves relevant historical cases for contextualization and save this case.

- **Draft Initial Article Task Output:** Records the initial draft content.
- **Integrate Additional Information Task Output:** Updates the draft with integrated sources and revisions.
- **Review and Authorize Publication Task Output:** Specifies the review status and comments from the Mayor.
- **Provide Social Media Feedback Task Output:** Details feedback posted on social media platforms, he can critize the mayor's decission.

4 Crew Interaction

4.1 CrewAI Flow

The Emergency Planner Flow implements a sophisticated flow-based approach for crew interactions:

- Emergency Services Crew processes initial call transcripts and produces assessments
- Medical Services and Firefighters Crews operate in parallel based on the assessment
- A Public Communication Crew activates only after both response teams have reported or during approval retries

As illustrated in Figure 5, the system uses logical operators (`and_`, `or_`) to create complex flow dependencies.

The flow is orchestrated using CrewAI's decorators, with complex triggering conditions:

```
@start()
def get_call_transcript():
    # Initial entry point

@listen(get_call_transcript)
def emergency_services():
    # Processes emergency call

@listen(emergency_services)
def firefighters():
    # Parallel response team

@listen(emergency_services)
def medical_services():
    # Parallel response team

@listen(or_(and_(firefighters, medical_services), "retry public
   communication"))
def public_communication():
    # Activates after both teams report or during retries

@listen("save full emergency report")
def save_full_emergency_report():
```

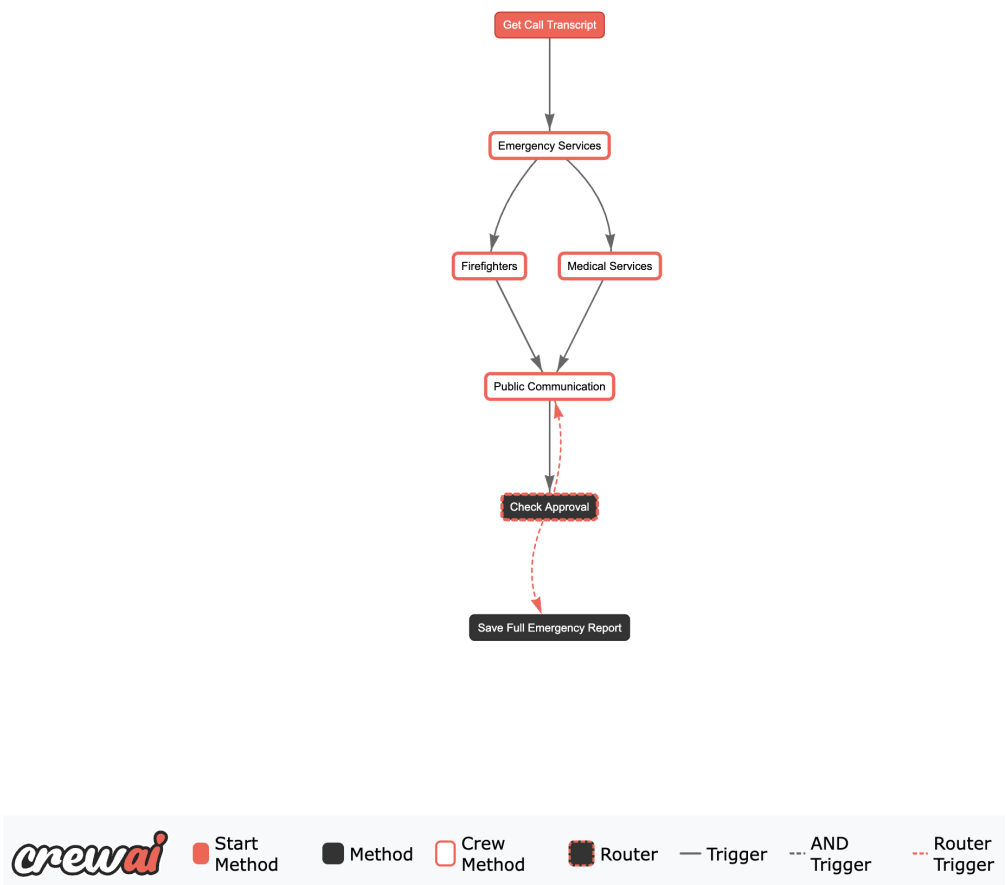

Figure 5: Emergency Response Flow showing parallel processing paths and conditional triggers

```
    # Generates final report with all crew responses
```

Listing 18: Key Flow Control Points in Emergency Response

4.2 State Management

The system maintains a centralized state using a Pydantic model that tracks all aspects of the emergency response:

```
class EmergencyPlannerState(BaseModel):
    call_transcript: Optional[str]
    call_assessment: Optional[CallAssessment]
    firefighters_response_report:
       Optional[FirefightersResponseReport]
    medical_response_report: Optional[MedicalResponseReport]
    public_communication_report: Optional[PublicCommunicationReport]
    mayor_approval_retry_count: int = 0
```

Listing 19: Emergency Planner State Model

This state model ensures:

- Complete tracking of the emergency call lifecycle

- Type-safe storage of crew assessments and reports
- Monitoring of mayoral approval attempts
- Optional fields to accommodate partial state updates

In addition to basic state management, the flow also performs basic data transformations, such as deciding whether to activate the medical services crew based on the call assessment, and extracting relevant data to pass to subsequent crews.

4.3 Router Implementation

The system employs a router specifically for managing public communications:

```
@router(public_communication)
def check_approval():
    if public_communication_report.mayor_approved:
        return "save full emergency report"
    elif mayor_approval_retry_count >=
       MAX_MAYOR_APPROVAL_RETRY_COUNT:
        return "save full emergency report"
    mayor_approval_retry_count += 1
    return "retry public communication"
```

Listing 20: Router Implementation for Public Communication Approval

4.4 Coordination Mechanism

The system implements a message-passing protocol using CrewAI's flow decorators:

- `@start()` marks the entry point for emergency call processing
- `@listen()` establishes dependencies between crew operations
- `@router()` handles conditional flow control

Parallel processing is achieved through independent `@listen` decorators, allowing medical and firefighter responses to operate concurrently. The flow concludes with a comprehensive emergency report that includes timestamps and summaries from all participating crews.

5 Conclusion

This report outlines the proposed cooperation and coordination mechanisms for the CrewAI MAS. Future work includes extending the interaction model to include additional agent types and testing the scalability of the system.