**1. System Architecture**

The code implements **7 specialized agents** that communicate via **ROS topics**:

| **Agent** | **Primary Responsibility** | **Key Enhancements** |
| --- | --- | --- |
| **SupervisingAgent** | Central coordinator, command interpretation | LLM for complex commands, RL data collection |
| **VisionAgent** | Face/obstacle detection, scene understanding | LLM for emotion/scene analysis |
| **AudioAgent** | Speech recognition and text-to-speech | Conversational LLM integration |
| **MovementAgent** | Leg/servo control for locomotion | LLM-optimized movement strategies |
| **GestureAgent** | Trunk/ear/tail/eye movements | Emotion-aware gestures |
| **TouchAgent** | Touch sensor interpretation | LLM for touch pattern responses |
| **RFIDAgent** | Mahout (handler) identification via RFID | Personalized interactions |

**2. Core Components**

**A. Data Structures**

* MahoutInfo: Tracks the elephant's handler (RFID, face recognition, voice).
* Command: Stores voice/gesture commands with metadata (source, confidence).
* SensorData: Aggregates touch/proximity/camera inputs.
* AgentContext: Short/long-term memory for LLM context (unique per agent).

**B. LLM Integration**

* Uses **Ollama** (local LLM) for:
  + Command interpretation (SupervisingAgent)
  + Scene description (VisionAgent)
  + Conversational responses (AudioAgent)
  + Adaptive movement (MovementAgent)
* **Caching**: LLM responses are cached to disk (LLM\_CACHE\_DIR) to avoid redundant queries.

**C. Reinforcement Learning (RL) Prep**

* **Data Collection**:
  + Each agent logs (state, action, reward, next\_state) tuples.
  + Saved to RL\_DATA\_DIR for future training.
* **Rewards**:
  + Positive for successful actions (e.g., complex\_actions = 1.0).
  + Negative for failures (e.g., unknown\_action = -0.1).

**3. Agent-Specific Enhancements**

**A. SupervisingAgent**

* **LLM-Powered Command Interpretation**:

python

def interpret\_command(self, command\_text: str) -> str:

prompt = f"Interpret this command for a robotic elephant: {command\_text}"

return get\_llm\_response(prompt, "supervising\_agent")

* **Complex Actions**: Sequences like perform\_dance() combine movements/gestures.
* **RL Data**: Logs rewards for actions (e.g., reward = 0.5 for movement).

**B. VisionAgent**

* **Scene Analysis**:

python

def analyze\_scene(self, frame):

prompt = "Describe this scene from the elephant's perspective..."

analysis = get\_llm\_response(prompt, "vision\_agent")

self.scene\_pub.publish(analysis) *# Sends JSON to ROS*

* **Emotion Detection**: Uses LLM to analyze facial expressions.

**C. AudioAgent**

* **Conversation Mode**:

python

def handle\_conversation(self, text):

prompt = f"Respond to this as a robotic elephant: {text}"

response = get\_llm\_response(prompt, "audio\_agent")

self.speak\_response(response)

**D. MovementAgent**

* **Adaptive Strategies**:

python

def optimize\_movement(self, command, environment):

prompt = f"Optimize {command} for environment: {environment}"

return get\_llm\_response(prompt, "movement\_agent") *# Returns speed/step adjustments*

**E. GestureAgent**

* **Emotion Adaptation**:

python

def adapt\_gesture(self, gesture, emotion):

prompt = f"Adjust {gesture} for emotion: {emotion}"

return get\_llm\_response(prompt, "gesture\_agent") *# Returns modified parameters*

**F. TouchAgent**

* **Intelligent Responses**:

python

def interpret\_touch(self, location, intensity):

prompt = f"Suggest response to touch at {location} (intensity: {intensity})"

return get\_llm\_response(prompt, "touch\_agent") *# e.g., "affectionate", "playful"*

**G. RFIDAgent**

* **Personalized Greetings**:

python

def personalize\_interaction(self, mahout\_id):

prompt = f"Generate greeting for mahout {mahout\_id}"

return get\_llm\_response(prompt, "rfid\_agent") *# Returns style/gestures*

**4. ROS Communication**

| **Topic** | **Publisher** | **Subscribers** | **Purpose** |
| --- | --- | --- | --- |
| /elephant/movement\_cmd | SupervisingAgent | MovementAgent | Commands like forward, turn\_left |
| /elephant/gesture\_cmd | SupervisingAgent | GestureAgent | Gestures like trunk\_raise |
| /elephant/mahout\_detected | Vision/RFID Agents | SupervisingAgent | Mahout presence/identity |
| /elephant/scene\_description | VisionAgent | SupervisingAgent | LLM-generated scene analysis |
| /elephant/conversation | AudioAgent | SupervisingAgent | Human-robot dialogue logs |

**5. Execution Flow**

1. **Startup**:
   * Each agent initializes its LLM, GPIO (for hardware), and ROS connections.
   * Example: MovementAgent sets up servo PWM signals.
2. **Main Loop**:
   * Agents process inputs (e.g., camera frames, voice commands).
   * LLMs generate context-aware responses.
   * Actions are executed (e.g., moving legs, flapping ears).
3. **RL Data Collection**:
   * After each action, the system logs:

python

save\_rl\_data("supervising\_agent", prev\_state, "forward", 0.5, next\_state)

**6. Key Innovations**

1. **Context-Aware LLMs**:
   * Each agent maintains its own memory (e.g., VisionAgent remembers recent scenes).
2. **Modularity**:
   * Agents can be run independently (e.g., python ai\_elephant\_agents.py vision).
3. **Hardware Integration**:
   * GPIO for servos, RFID reader, and touch sensors.
4. **Scalability**:
   * New agents can be added without disrupting others.

**7. Example Scenario**

**Command**: "Hey elephant, dance for me!"

1. **AudioAgent** converts speech to text, publishes to /elephant/command\_received.
2. **SupervisingAgent** uses LLM to interpret "dance" as perform\_dance().
3. **MovementAgent** and **GestureAgent** coordinate:
   * Legs move in rhythm (turn\_left, turn\_right).
   * Ears flap, tail wags.
4. **RL Data**: The successful dance logs a reward of 1.0.

**8. Future Work**

* **Train RL Policies**: Use collected data to optimize actions.
* **Multi-Modal LLMs**: Add image/video understanding.
* **Edge Deployment**: Compress LLMs for Raspberry Pi.

This system combines **robotics, LLMs, and RL** to create an elephant that learns and interacts naturally!