# Step-by-Step Explanation of LangGraph-based Epic Evaluation System

**Overview**

This script implements an advanced, dynamic multi-agent system using **LangGraph** to evaluate Agile Epic descriptions. It leverages Google's **Gemini 1.5 Flash** model to analyze epic elements, provide quality ratings, and generate detailed, actionable feedback for improvements. The system is designed with conditional workflows and iterative processing to ensure thorough evaluation and refinement.

**1. Environment Setup**

* **.env file:** Securely stores the GOOGLE\_API\_KEY.
* **load\_dotenv():** Loads environment variables from the **.env file** into the Python script.
* **API Key:** Essential for authenticating and accessing the **Gemini model API**.
* **Gemini Model Initialization:** ChatGoogleGenerativeAI(model="gemini-1.5-flash", google\_api\_key=api\_key) initializes the language model for all **AI** interactions.

**2. Libraries Used**

* **os:** For interacting with the operating system, like creating directories (os.makedirs) and accessing environment variables (os.getenv).
* **dotenv:** To load environment variables from **.env files**.
* **json:** For working with **JSON** data (reading, writing, parsing).
* **time:** Used for adding delays (time.sleep()) to manage **API rate limits** and avoid overwhelming the model.
* **datetime:** For generating timestamped filenames for **evaluation reports**.
* **typing (Optional, List, Dict):** Used for **type hinting** to improve code readability and maintainability.
* **pydantic (BaseModel):** For defining data models (**ElementEvaluation**, **EpicState**) to enforce structured data and ensure data integrity.
* **langchain\_google\_genai (ChatGoogleGenerativeAI):** The **LangChain** integration for Google's **Generative AI models**.
* **langchain.prompts (PromptTemplate):** To create reusable and structured **prompts** for **LLM** interactions.
* **langchain.output\_parsers (StructuredOutputParser, ResponseSchema):** To define expected output formats from the **LLM** and parse the responses into **structured data**.
* **langgraph.graph (StateGraph, END):** The core framework used to build and manage the **multi-agent workflow** as a graph.

**3. Pydantic Models for State Management** The system uses **Pydantic models** to define and manage its state and data structures:

* **ElementEvaluation:** Represents the evaluation of a single epic element.
  + **element:** The name of the epic element (e.g., "Title", "Problem Statement").
  + **quality:** The quality rating (e.g., "HIGH", "MEDIUM", "LOW", "ERROR").
  + **explanation:** A justification for the quality rating.
  + **recommendations:** General suggestions for improvement.
  + **feedback** (Optional): Detailed, actionable steps for improvement, generated if the quality is "LOW".
* **EpicState:** Manages the overall state of an epic as it flows through the graph.
  + **raw\_text:** The original epic description text.
  + **parsed** (Optional[Dict]): The structured elements extracted from the epic.
  + **evaluations** (List[ElementEvaluation]): A list of all evaluations performed for the epic.
  + **current\_element** (Optional[str]): The name of the element currently being processed.
  + **current\_content** (Optional[str]): The content of the element currently being processed.
  + **refinement\_needed** (Optional[bool]): Flag indicating if the last evaluated element needs detailed refinement.
  + **done:** Flag indicating if all elements of the epic have been processed.
  + **epic\_index** (Optional[int]): The index of the epic being processed, used for file naming.

**4. Output Parsers** **StructuredOutputParser** and **ResponseSchema** are used to define the expected **JSON** format from the **LLM**, ensuring that the responses are structured and can be reliably processed by the script.

* **parser\_schema:** Defines the structure for extracting epic elements (Title, Problem Statement, etc.).
* **eval\_schema:** Defines the structure for evaluation results (quality, explanation, recommendations).

**5. QUALITY\_STANDARDS** A multiline string **QUALITY\_STANDARDS** is defined within **element\_evaluator**. This string provides the specific criteria and examples for what constitutes "HIGH", "MEDIUM", and "LOW" quality for each epic element (Title, Problem Statement, Product Outcome & Instrumentation, Requirements - User Stories, Non-Functional Requirements). These standards guide the **Gemini model** during the evaluation process.

**6. LangGraph Nodes (Modular Agents)** The core logic is divided into distinct nodes, acting as specialized agents within the **LangGraph framework**:

**a. parser\_node (Epic Parser Agent)**

* **Function:** This node is the **entry point** for each epic. It takes the **raw\_text** of an epic, prompts the **Gemini model** to extract and organize key elements (like Title, Problem Statement, User Stories, etc.) into a **structured JSON format**.
* **Error Handling:** It includes a **try-except** block to handle potential **JSONDecodeError** and uses parser.parse() as a fallback. It also ensures all expected keys are present, filling empty strings if an element is not found in the raw text.
* **Output:** Updates the **parsed** field in the **EpicState** with the extracted elements.
* **Transition:** Passes the state to the **element\_router**.

**b. element\_router (Conditional Logic / Router Node)**

* **Function:** This node acts as the **central director** of the workflow. It determines which element needs to be evaluated next or if all elements for the current epic have been processed.
* **Logic:** It compares the set of all **parsed elements** with the set of already **evaluated elements**.
* **Output:** Sets **current\_element** and **current\_content** in **EpicState** if there are more elements to evaluate. If all elements are evaluated, it sets **done** to **True**.
* **Transitions:**
  + If **state.done** is **True**, it transitions to **aggregate\_node**.
  + Otherwise, it transitions to **element\_evaluator**.

**c. element\_evaluator (Specialized Element Evaluator Agent)**

* **Function:** This node is responsible for evaluating a single **current\_element** against the **QUALITY\_STANDARDS**. It prompts the **Gemini model** to provide a **quality** rating, an **explanation**, and **recommendations**.
* **Guardrails:**
  + **Retry Logic:** Implements **MAX\_RETRIES** to re-attempt **LLM invocation** in case of transient errors or invalid responses.
  + **Output Validation:** Validates the **quality** score against **VALID\_QUALITIES** ("HIGH", "MEDIUM", "LOW", "ERROR") and checks if the **explanation** meets a **MIN\_EXPLANATION\_LENGTH**. If validation fails, it retries or marks as "ERROR".
* **Output:** Appends a new **ElementEvaluation** object to the **evaluations** list in **EpicState**. It also sets **refinement\_needed** to **True** if the evaluated **quality** is "LOW".
* **Transitions:**
  + If **state.refinement\_needed** is **True** (meaning the **quality** was "LOW"), it transitions to **refinement\_node**.
  + Otherwise (quality is "HIGH" or "MEDIUM"), it transitions back to **element\_router** to pick the next element.

**d. refinement\_node (Refinement/Detailed Feedback Agent)**

* **Function:** This node is **conditionally activated** when an element receives a "LOW" quality rating. It prompts the **Gemini model** again to generate more detailed, actionable, and concrete steps or examples for improving that specific element.
* **Output:** Updates the **feedback** field of the last **ElementEvaluation** object in **EpicState** with the refined suggestions. It then resets **refinement\_needed** to **False**.
* **Transition:** Always transitions back to **element\_router** to ensure the overall workflow continues to process other elements.

**e. aggregate\_node (Report Aggregator Agent)**

* **Function:** This node is the **final step** for each epic. It collects all **ElementEvaluation** objects for the current epic from **EpicState** and compiles them into a single, comprehensive **JSON report**.
* **File Handling:**
  + Creates an **evaluation\_results** directory if it doesn't already exist.
  + Generates a unique filename for each epic's report using the **epic\_index** and a **YYYYMMDD\_HHMMSS** timestamp (e.g., epic\_evaluation\_1\_20240610\_110227.json).
  + Saves the **JSON report** to this file.
* **Output:** Prints the final **JSON report** to the console and saves it to a file.
* **Transition:** Transitions to **END**, signifying the completion of processing for the current epic.

**7. LangGraph Workflow Configuration**

* **graph = StateGraph(EpicState):** Initializes the graph with **EpicState** to manage the shared data.
* **graph.add\_node(...):** Each of the functions (**parser\_node**, **element\_router**, **element\_evaluator**, **refinement\_node**, **aggregate\_node**) is added as a named node to the graph.
* **graph.set\_entry\_point("parser"):** Defines "parser" as the **starting point** of the workflow.
* **graph.add\_edge(...):** Defines direct transitions between nodes (e.g., **parser** to **router**).
* **graph.add\_conditional\_edges(...):** Defines dynamic transitions based on the state.
  + **router** decides whether to go to **aggregate** (if **done**) or **evaluator**.
  + **evaluator** decides whether to go to **refiner** (if **refinement\_needed**) or **router**.
* **graph.add\_edge("aggregate", END):** Explicitly marks the **end** of an epic's processing flow.
* **epic\_flow = graph.compile():** Compiles the defined graph into an executable workflow.

**8. Execution (if \_\_name\_\_ == "\_\_main\_\_":)**

* **sample\_epics:** A list containing **sample epic texts** is defined for demonstration.
* **Looping:** The script iterates through each **sample\_epic**.
* **epic\_flow.invoke(EpicState(raw\_text=epic, epic\_index=i)):** For each epic, it invokes the **LangGraph workflow**, passing the **raw\_text** and an **epic\_index** to the initial **EpicState**.
* **time.sleep(4):** A pause is added after processing each epic to manage **API quotas** and prevent hitting rate limits during batch processing.
* **Output:** As each epic is processed, its final evaluation report is printed to the console and saved as a separate timestamped **JSON file** in the **evaluation\_results** directory.