**Overview of LangGraph Workflow**

The project implements an advanced disease prediction system using a multi-agent architecture, integrating a prediction model, a vector database for external knowledge, and a transparent, explainable workflow. The backend is built with Django, and the multi-agent logic is organized in a langgraph\_agents subfolder containing several key files: tools.py, state.py, nodes.py, graphs.py, config.py, and api\_integration.py. Below is a breakdown of each file's likely role and how they fit together:

**File-by-File Explanation**

1. tools.py

Purpose: Defines utility functions and "tools" for agents to use during the diagnostic process. Defines the tools that the agents can use (e.g., searching medical knowledge, matching symptoms to diseases, getting disease precautions).

`create\_medical\_tools`: Initializes and returns the tools for the agents.

Importance: Provides the agents with access to external knowledge and functionality.

Role in Workflow: Provides callable functions (e.g., querying the vector database, fetching medical facts) that are wrapped and exposed to the graph as ToolNode, allowing agents to access structured, external knowledge or perform specific actions.

2. state.py

Purpose: Defines the DiagnosticState class or structure.

Role in Workflow: Specifies the shape and fields of the state object that flows through the graph. This includes patient info, symptoms, predictions, questions, answers, reasoning steps, and other metadata that must persist and evolve as the workflow progresses.

3. nodes.py

Purpose: Contains the implementations of the various agent nodes (e.g., orchestrator, questioning, validation). Each node performs a specific task (e.g., generating questions, refining predictions, generating explanations).

Role in Workflow: Each function or class here represents a step in the diagnostic reasoning process, such as:

* Deciding the next action (orchestrator)
* Generating follow-up questions (questioning)
* Handling human input (human\_input)
* Integrating responses (response\_integration)
* Validating and refining predictions (validation, refinement)
* Generating explanations (explanation)
* Evaluating if further questioning is needed (evaluator)

Below is a breakdown of each major component and function, explained simply:

**Class: MedicalAgentNodes**

This class contains all the agent "nodes" (steps) used in your disease diagnostic reasoning process. When you create an instance, you provide:

* A vector database (for external knowledge retrieval)
* A Gemini API key (for Google Generative AI model access)

**Initialization (\_\_init\_\_)**

* Sets up the language model (LLM) using Gemini.
* Binds medical tools (from your vector DB) to the LLM for enhanced reasoning.
* Prepares the LLM for both direct predictions and tool-augmented tasks[1](https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/6007531/1e745c91-8bce-4833-a64c-2e78fabb55fb/paste.txt).

**Agent Nodes (Methods)**

**1. Orchestrator Node (orchestrator\_node)**

* **Role:** Acts as the "project manager" for the diagnostic workflow.
* **What it does:**
  + Analyzes initial disease predictions and selected symptoms.
  + Decides if more information is needed (e.g., if confidence is too low).
  + Sets workflow parameters and adds a reasoning step for transparency.
  + Updates the system state with its analysis and whether more questions are needed.
* **Transparency:** Adds its reasoning as a step in the state for later explanation[1](https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/6007531/1e745c91-8bce-4833-a64c-2e78fabb55fb/paste.txt).

**2. Questioning Node (questioning\_node)**

* **Role:** Generates clarifying questions to refine the diagnosis.
* **What it does:**
  + Checks how many more questions can be asked (enforces a limit).
  + Prompts the LLM to generate high-priority yes/no questions that help differentiate between top predicted diseases.
  + Parses the LLM's output, handling potential formatting issues robustly.
  + If parsing fails, generates fallback questions programmatically.
  + Updates the state with the generated questions and reasoning.
* **Transparency:** Logs reasoning about question generation and includes the raw LLM output in the state[1](https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/6007531/1e745c91-8bce-4833-a64c-2e78fabb55fb/paste.txt).

**3. Human Input Node (human\_input\_node)**

* **Role:** Handles interaction with the user (collects answers to clarifying questions).
* **What it does:**
  + Checks if user responses are available.
  + If yes, updates the state to indicate responses were received.
  + If not, waits for input or skips if there are no questions to ask.
* **Transparency:** Clearly tracks the workflow step and logs the number of questions and responses[1](https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/6007531/1e745c91-8bce-4833-a64c-2e78fabb55fb/paste.txt).

**4. Response Integration Node (response\_integration\_node)**

* **Role:** Updates the symptom profile based on user answers.
* **What it does:**
  + Prompts the LLM to analyze which symptoms to add/remove based on user responses.
  + Parses the LLM's output to update the symptom list and logs the analysis.
  + Handles parsing errors gracefully and logs any issues.
* **Transparency:** Adds a reasoning step summarizing how responses changed the symptom profile[1](https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/6007531/1e745c91-8bce-4833-a64c-2e78fabb55fb/paste.txt).

**5. Refinement Node (refinement\_node)**

* **Role:** Refines and re-ranks disease predictions after new information is integrated.
* **What it does:**
  + Prompts the LLM to update disease probabilities and confidence levels based on updated symptoms.
  + Requests explanations for any changes in ranking.
  + Uses medical tools for validation.
  + Updates the state with new predictions and explanations.
* **Transparency:** Ensures every change is accompanied by an explanation, supporting traceability[1](https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/6007531/1e745c91-8bce-4833-a64c-2e78fabb55fb/paste.txt).

6. **Validation Node**

**Purpose**

* The validation node checks the plausibility and medical correctness of the current predictions and reasoning steps.
* It acts as a “medical fact-checker,” ensuring that the predictions align with established medical knowledge.

**What it typically does**

* **Cross-references** predicted diseases and symptoms with your vector database or external medical sources.
* **Flags inconsistencies** or missing critical symptoms for a given disease.
* **May adjust confidence scores** or trigger a request for more information if a prediction is not well-supported.

**Transparency:** Every validation result and its reasoning are logged for review.

7. **Explanation Node**

**Purpose**

* The explanation node synthesizes a clear, human-readable summary of why the system made its predictions.
* It’s responsible for the “explainability” requirement—making sure users understand the reasoning behind the output.

**What it typically does**

* **Aggregates reasoning steps** from previous nodes.
* **Generates a narrative** or bullet-point explanation for each prediction, referencing symptoms, user answers, and validation checks.
* **Formats output** for display in your React frontend.

**Transparency:** The explanations reference all prior reasoning and are available for user inspection.

8. **Evaluator Node**

**Purpose**

* The evaluator node assesses the overall quality and reliability of the diagnostic process.
* It can be used for both internal QA and user-facing feedback (e.g., “How confident is the system in this diagnosis?”).

**What it typically does**

* **Reviews all steps** (predictions, validation, explanations).
* **Assigns an overall confidence score** or “trust level” to the diagnostic session.
* **May flag cases** for human review if confidence is low or inconsistencies are found.

**Transparency:** The evaluation and its reasoning are stored in the state for auditability.

|  |  |  |  |
| --- | --- | --- | --- |
| **Node Name** | **Role in System** | **Key Actions** | **Transparency Mechanism** |
| Orchestrator Node | Coordinates diagnostic workflow | Analyzes predictions, sets next steps | Adds reasoning step |
| Questioning Node | Generates clarifying questions | Prompts LLM, parses/generates questions, updates state | Logs reasoning and LLM output |
| Human Input Node | Handles user interaction | Waits for/responds to user answers | Tracks workflow step |
| Response Integration Node | Updates symptom profile from user responses | Prompts LLM, parses updates, logs analysis | Adds reasoning step |
| Refinement Node | Refines predictions after new info | Updates probabilities/confidence, requests explanations | Ensures explanations for changes |
| Validation Node | Checks medical correctness | Validation results, flags inconsistencies | Logs checks and outcomes |
| Explanation Node | Generates user-facing explanations | Narrative or bullet-point explanations | References all reasoning |
| Evaluator Node | Assesses overall diagnostic quality | Session confidence/trust score | Logs evaluation reasoning |

4. graph.py

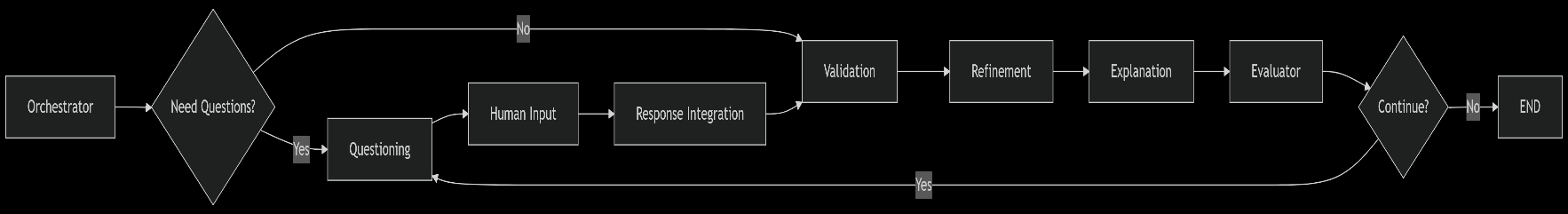
Purpose: Defines the overall diagnostic workflow as a LangGraph state machine. It sets up how different agents and tools interact, manages state and ensures the system is transparent and interactive.

Key Features:

* Defines the state machine with nodes and conditional edges
* Manages session persistence with SQLite checkpoints
* Handles interruption points for human input
* Formats responses for questions and final results

Role in Workflow:

* Initializes all nodes and tools.
* Sets up the workflow graph, specifying the flow of information and decision points (edges and conditional transitions).
* Handles persistence (using SQLite via SqliteSaver) so sessions can be paused and resumed.
* Provides methods to start a diagnosis (run\_diagnosis) and continue with user answers (continue\_with\_answer).
* Integrates tracing (LangSmith) for transparency and debugging.



Code Explanation

**Class: MedicalDiagnosticGraph**

This is the main class that sets up and runs the diagnostic workflow.

**Initialization (\_\_init\_\_)**

* **Inputs:** References to the vector database, Gemini API key, and path for storing checkpoints.
* **Nodes & Tools:** Instantiates agent nodes (MedicalAgentNodes) and medical tools.
* **LangSmith Tracing:** Optionally enables tracing for debugging and transparency.
* **Checkpointer:** Sets up SQLite-based persistence so sessions can be paused/resumed.
* **Graph Building:** Calls \_build\_graph() to construct the workflow.

**Checkpointer (\_create\_checkpointer)**

* Ensures the checkpoint database directory exists.
* Opens a SQLite connection for session persistence.
* Returns a SqliteSaver to manage saving and loading workflow state.

**Workflow Construction (\_build\_graph)**

* **Creates a StateGraph** using the DiagnosticState (your state model).
* **Adds nodes** for each step:
  + orchestrator: Decides next steps.
  + questioning: Generates follow-up questions.
  + human\_input: Waits for user answers.
  + response\_integration: Incorporates user answers.
  + validation: Checks prediction quality.
  + refinement: Refines predictions.
  + explanation: Generates explanations.
  + evaluator: Decides if more questions are needed.
  + tools: Provides access to external medical tools.
* **Defines workflow transitions**:
  + Entry point is the orchestrator.
  + Conditional transitions (e.g., whether to ask more questions).
  + Parallel and sequential steps (e.g., validation and refinement).
  + Interrupts for human input.
* **Compiles the graph** with or without persistence.

**Decision Functions**

* \_should\_ask\_questions: Checks if the state says more questions are needed.
* \_should\_continue: Checks if more questions should be asked or the workflow should end.

**Main Workflow Execution (run\_diagnosis)**

* **Inputs:** Symptoms, patient info, initial predictions, session ID, max questions.
* **Initializes state** with all relevant info and metadata for traceability.
* **Runs the workflow** step by step, streaming state updates.
* **Handles human input interrupts:** If the system needs more info from the user, it returns a formatted set of questions.
* **Handles normal completion:** Formats and returns the final results.
* **Error handling:** Catches and reports any errors, returning partial results if needed.

**Formatting Question Responses (\_format\_question\_response)**

* Extracts and formats clarifying questions for the frontend.
* Ensures each question has a clear structure (ID, text, type, etc.).
* Handles cases where questions are missing by providing a default.
* Returns a response indicating the system is waiting for user input.

**Continuing After Human Input (continue\_with\_answer)**

* Loads the current session state using the checkpointer.
* Updates state with the user's answers.
* Continues the workflow from the point it was paused.

**How This Enables Transparent, Interactive Diagnostics**

* **Step-by-step Reasoning:** Each node in the graph represents a logical step, making the workflow easy to trace and explain.
* **Conditional Logic:** The system decides dynamically whether to ask more questions or proceed, based on the current state.
* **Persistence:** Users can pause and resume sessions, with all reasoning steps saved.
* **Human-in-the-Loop:** The system can interrupt itself to ask users clarifying questions, then seamlessly continue.
* **Traceability:** LangSmith integration enables full tracing of all steps and decisions for transparency.
* **Structured Explanations:** The explanation node ensures that every output can be accompanied by a clear, structured explanation.

|  |  |
| --- | --- |
| **Function / Method** | **Purpose** |
| \_\_init\_\_ | Initializes workflow, nodes, tools, tracing, and persistence |
| \_create\_checkpointer | Sets up SQLite-based session persistence |
| \_build\_graph | Defines the diagnostic workflow as a graph of steps and transitions |
| \_should\_ask\_questions | Decides if more clarifying questions are needed |
| \_should\_continue | Decides if the workflow should continue or end |
| run\_diagnosis | Runs the diagnostic process, handling state and interrupts |
| \_format\_question\_response | Formats questions for user input, ensuring clarity and structure |
| continue\_with\_answer | Updates state with user answers and resumes the workflow |

5. config.py

Purpose: Stores configuration settings, such as API keys, database paths, and feature toggles.

Role in Workflow: Centralizes all environment-dependent or sensitive settings, allowing the rest of the code to remain clean and consistent.

6. api\_integration.py

Purpose: Handles communication with external APIs (e.g., LLMs, medical data providers).

Role in Workflow: Abstracts away the details of API calls, so nodes can simply call functions to get predictions or additional data.

Workflow Explanation

1. Initial Prediction

The model makes an initial prediction based on user symptoms and patient info.

These results are sent to the multi-agent system (the LangGraph workflow).

2. Orchestration

The orchestrator node decides whether more clarifying questions are needed or if the workflow can proceed directly to validation.

This decision uses the current state, especially the needs\_more\_questions flag.

3. Questioning and Human Input

If more information is needed, the questioning node generates clarifying questions.

The workflow transitions to human\_input, which pauses and waits for user responses (interrupt point).

The system formats and returns the questions to the frontend for user input.

4. Response Integration

Once answers are received, they're integrated into the state by the response\_integration node.

The system updates symptoms, reasoning steps, and possibly the prediction.

5. Validation and Refinement

The validation node checks the updated predictions for consistency and confidence.

The refinement node may adjust predictions or reasoning based on new data.

6. Explanation

The explanation node compiles a transparent, step-by-step rationale for the current prediction, referencing both the model's output and the multi-agent reasoning process.

7. Evaluation and Looping

The evaluator node decides if more questioning is needed (based on question count, confidence, etc.).

If so, the workflow loops back to questioning; otherwise, it proceeds to completion.

8. Completion

The final state, including the diagnosis, reasoning steps, and explanations, is returned to the frontend.

