

ASSIGNMENT 4:
NORTHWESTERN MEMORIAL HEALTHCARE AGENT

Kevin Geidel
MSDS 442: AI Agent Design & Development
Northwestern University
May 25, 2025

Requirements 1 and 2: Graph the agent with LangChain/LangGraph

The construction of the agent and accompanying graph begins with the creation of the elements (classes and functions) that serve as nodes and edges in our graph. This begins as early as cell 2 (see appendix) with the creation of the `InquiryState` class that tracks the current values and message history for the agent cluster. The actual assembly of the graph itself occurs in cell 13. However the components are instantiated above that. Following the logic in cell 13 we first instantiate an empty graph:

```
builder = StateGraph(InquiryState)
```

The builder object has `add_node` and `add_edge` methods that allow us to assemble the components created in cells 2-12. We use `add_conditional_edges` to inform the graph that the next node will be determined dynamically, depending on agent/user interactions. The output is displayed graphically in cell 14 (reproduced in figure 1 below.)

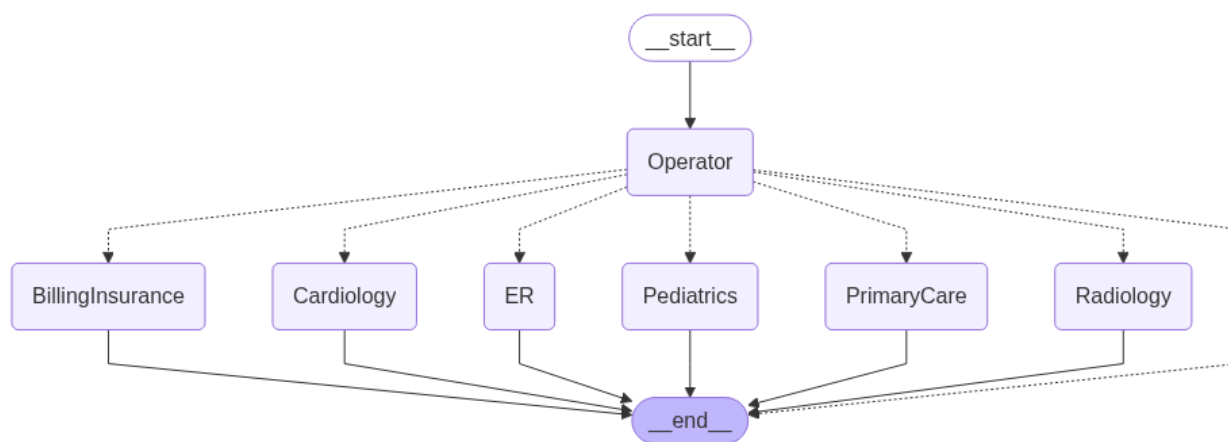


Figure 1: Graphical depiction of the Northwestern Memorial Healthcare Agent.

We can compare the workflow in figure 1 to the flowchart of desired functionality provided by Dr. Bader in the assignment requirements (figure 2.) Each desired agent is represented as a node in the LangGraph. Edges represent the state being transferred from one agent to the next.

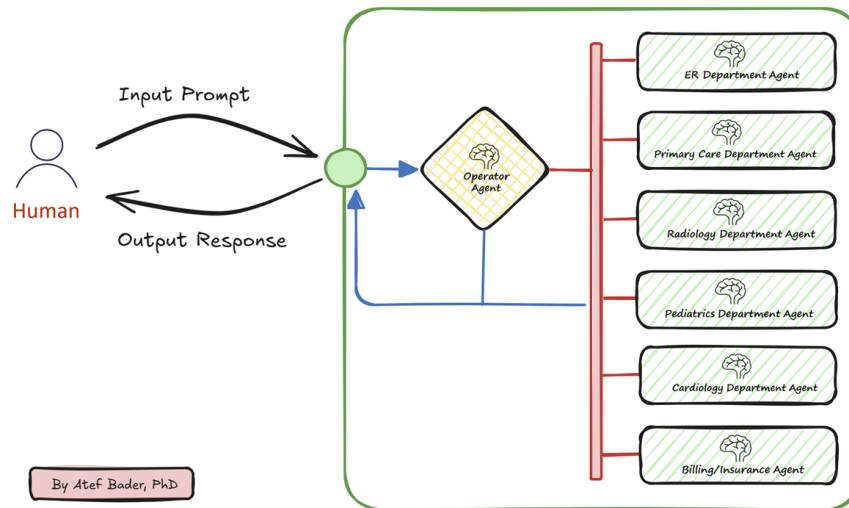


Figure 2: The desired workflow as provided by the assignment objectives.

Requirement 3: The operator agent

An operator agent acts as a router to direct queries to and from the various department agents. The operator is constructed in cell 5 (reproduced in figure 3 with line numbers.) Lines 2 and 3 extract the current inquiry and message history respectively. Lines 5-13 check for any termination conditions and end the conversation gracefully if present.

In order to accommodate multi-turn conversations the operator must be able to detect and respond accordingly to existing chat history. The conditional in line 16, `if state.get('referring_node') != "Operator"` and `state.get('next_node')`, will evaluate to True if the department agents have already interacted with the user at least once (`next_node` is not set until initially.) The five most recent messages are extracted from the state in line 17 (limit of five to avoid token overflows.) In lines 18-29 the operator invokes the LLM with a prompt crafted to evaluate the chat history and determine if the latest user query is part of an on going conversation (i.e. a follow-up question, an additional question or providing requested information.) If the LLM determines this is the case the inquiry is forwarded back to the referring node/department. This query can also decide that the latest inquiry is best served by a different department and 'transfer' the inquiry to it (instead of the referring node.)

```

1 def operator_router(state):
2     inquiry = state['inquiry'].lower()
3     messages = state.get("messages", [])
4
5     # Check for end of conversation
6     if inquiry in ['q', 'quit']:
7         return {
8             "inquiry": state["inquiry"],
9             "referring_node": "Operator",
10            "next_node": END,
11            "response": "Goodbye! Thank you for contacting Northwestern Memorial!",
12            "messages": messages + [HumanMessage(content=inquiry), SystemMessage(content="Conversation ended by user.")]
13        }
14
15    # Check for an ongoing conversation
16    if state.get('referring_node') != "Operator" and state.get('next_node'):
17        history = "\n".join([f"{msg.type}: {msg.content}" for msg in messages[:5]])
18        query = f"""Given the conversation history and the new inquiry: '{inquiry}', determine if this is a follow-up question related to the previous
19        department ({state['referring_node']}) or a new topic. Return 'continue' if it's a follow-up, or classify the intent for a new topic.
20        Possible intent values: Greeting, GeneralInquiry, ER, Radiology, PrimaryCare, Cardiology, Pediatrics, BillingInsurance
21
22        Conversation history:
23        {history}
24        """
25        messages_for_intent = [
26            SystemMessage(content="You are a helpful assistant tasked with classifying the intent of a user's query or detecting follow-ups."),
27            HumanMessage(content=[{"type": 'text', 'text': query}])
28        ]
29        response = llm.invoke(messages_for_intent)
30        intent = response.content.strip()
31        if intent == 'continue':
32            return {
33                "inquiry": state["inquiry"],
34                "referring_node": "Operator",
35                "next_node": state['referring_node'],
36                "response": f"Continuing with the {state['referring_node']} department.",
37                "messages": messages + [HumanMessage(content=inquiry)]
38            }
39
40    # This is a new conversation. Have the operator decide how to route.
41    query = f"""Classify the user's intents based on the following input: '{state['inquiry']}'
42    List of possible intent values: Greeting, GeneralInquiry, ER, Radiology, PrimaryCare, Cardiology, Pediatrics, BillingInsurance
43    Return only the intent value of the inquiry identified with no extra text or characters"""
44    messages = [
45        SystemMessage(content="You are a helpful assistant tasked with classifying the intent of user's inquiry"),
46        HumanMessage(content=[{"type": "text", "text": query}]),
47    ]
48    response = llm.invoke(messages)
49    intent = response.content.strip()
50    response_lower = intent.lower()
51
52    if "greeting" in response_lower:
53        response = "Hello there, This is Northwestern Memorial Hospital, How can I assist you today?"
54        next_node = END
55    elif "generalinquiry" in response_lower:
56        response = "For general information about nearby parking, hotels and restaurants, please visit https://www.nm.org/ and navigate to Patients & Visitors link "
57        next_node = END
58    else:
59        response = f"Let me forward your query to our {intent} agent."
60        next_node = intent
61
62    return {
63        "inquiry": state["inquiry"],
64        "referring_node": "Operator",
65        "next_node": next_node,
66        "response": response,
67    }
68

```

Figure 3: The operator/router is instantiated in cell 5.

By line 39 we are on the other side of the existing conversation conditional and we can be confident that the inquiry marks a new conversation. Lines 40-50 repeat a similar query to the LLM- classifying user intent for the purposes of routing the inquiry. The difference here is there prompt does not have to consider if the inquiry is related to historic messages at all. Lines 52-57 set some hard coded responses and routes for some specific intents but, in lines 62-67, the inquiry is routed to the appropriate department specific agent.

Requirement 4: Department specific agents

Cells 7-12 instantiate functions that represent the six agents and their respective departments. The reader will likely note these six functions simply return the output of a common function, passed department specific arguments. The function, `department_specific_agent`, is an abstracted agent that performs the same steps for each department agent. These steps include loading conversation history, checking for (and reacting to) termination conditions, determining if this is an ongoing or new conversation, loading the department's knowledge base into context for new conversations, checking if the user's issue has been resolved and forwarding the state to the appropriate next node. The base department agent is defined in cell 6 and reproduced, with line numbers, in figure 4.

```

1 def department_specific_agent(state, department_node_name, knowledge_base_filename):
2     # Handle inquiries related to the passed department
3     inquiry = state['inquiry'].lower()
4     messages = state.get("messages", [])
5
6     # Check for end of conversation
7     if inquiry in ['q', 'quit']:
8         return {
9             "inquiry": state["inquiry"],
10            "referring_node": department_node_name,
11            "next_node": END,
12            "response": f"Goodbye! Thank you for contacting {department_node_name} at Northwestern Memorial!",
13            "messages": messages + [HumanMessage(content=inquiry), SystemMessage(content="Conversation ended by user.")]
14        }
15
16     if state['referring_node'] == 'Operator':
17         # This is first pass at the department agent, include the system message
18         messages += [get_system_message_for_agent(knowledge_base_filename)] + [get_human_message_for_agent(state)]
19     else:
20         # This is an ongoing conversation. Just append new inquiry
21         messages += [get_human_message_for_agent(state)]
22
23     response = llm.invoke(messages)
24     formatted_response = f"{department_node_name}:: " + response.content.strip()
25
26     # Check if conversation is over (next_node=END) or not (next_node=same department)
27     completion_check = llm.invoke([
28         SystemMessage(content=f"Determine if the user's inquiry is fully resolved based on the response: '{response.content}'.\n\nReturn 'complete' if resolved, 'continue' if further interaction is needed."),
29         HumanMessage(content=[{'type': 'text', 'text': f'Response: {response.content}'}])
30     ])
31     next_node = END if completion_check.content.strip() == 'continue' else department_node_name
32
33     return {
34         "input": state["inquiry"],
35         "referring_node": department_node_name,
36         "next_node": next_node,
37         "response": formatted_response,
38         "messages": messages + [SystemMessage(content=formatted_response)]
39     }

```

Figure 4: The abstracted department agent is instantiated in cell 6.

Lines 1-14 are similar to those of the operator- extracting what is needed from the current state and handling termination conditions (if detected.) This could likely be abstracted as well since the only differences related to specific hard coded messages to customize the experience (i.e. ‘Thank you for contacting the Cardiology Department!’ vs ‘Come again soon!’) Lines 16-21 load the latest (or first) user inquiry into messages ahead of invoking the LLM. The conditional is required to determine if the workflow has

been through this department yet. If the referring node is the operator then this is the first pass through the department. The department specific `SystemMessage` (which is created dynamically from utility functions defined in cell 4) is injected into the conversation ahead of the `UserMessage`. If the referring node is this department this step is unnecessary and the current inquiry appended only.

The actual call to the LLM takes place in lines 23 and 24. Before the agent can properly route the inquiry to the next node it must first determine if the conversation is over (in which case `next_node` is set to `END`) or if the user's request is not resolved (and `next_node` is set back to this department.) This check is accomplished with an LLM query created in lines 27-31. The prompt is simple and asks the LLM return 'continue' if the conversation is not adequately resolved.

Requirement 5: Classifying intent

LLM queries are used to determine user intent in three different places. Two occur within the operator agent. One query is for classifying a new inquiry so that it can be routed to the appropriate department agent. This can be found in lines 40-48 of cell 5 (see figure 3.) The second query that the operator agent can use to infer user intent is for existing conversations. The LLM (lines 18-29 of cell 5 in figure 3) is tasked with deciding if the latest inquiry must go back to the referring node (to continue an ongoing conversation) or be routed to a new department.

The third instance of the agents using the LLM to classify intent happens in the respective departments. The LLM is used to decide if the conversation is over or needs to be routed back around for a multi-turn conversation. This is implemented in lines 26-31 of cell 6 (see figure 4.)

Requirement 6: Knowledge base data

The agents instruct the LLM to restrict answers to the information in the department's respective knowledge base. The knowledge bases consist of six JSON files (one for each department) that contain a number of question ('inquiry') and answer ('response') entries. Some of these entries were provided with the assignment. I used Grok (<https://grok.com/>) to convert the provided questions and answers into the desired JSON format (figure 5.)

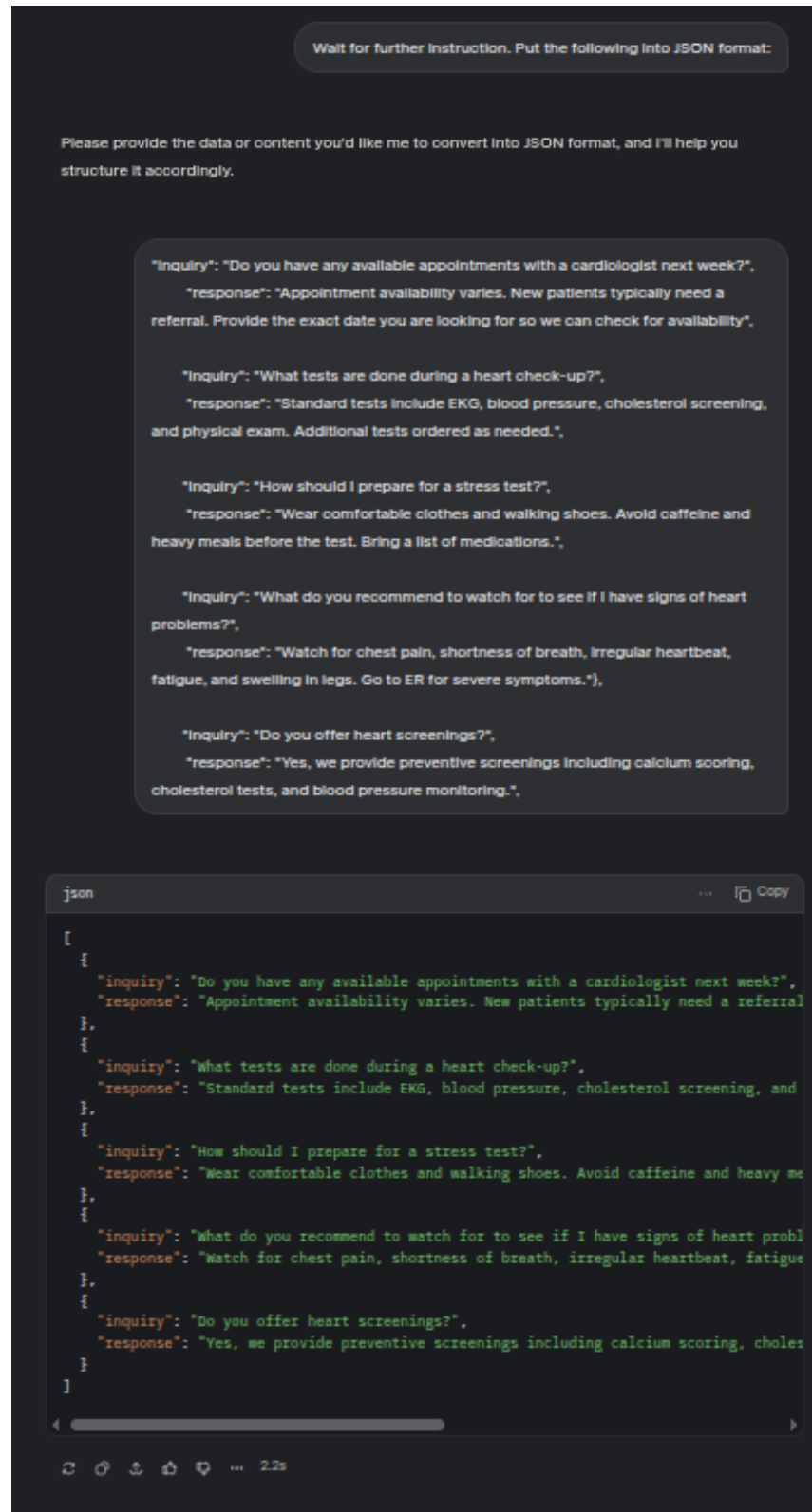


Figure 5: The commercial AI, Grok, converting knowledge base entries into a JSON object.

Grok was also used to generate new questions. Figure 6 shows how questions were created for the Pediatrics department and, utilizing Grok's multi-turn capability, obtaining the same JSON schema for the Emergency Department questions as well. This process was repeated until all six departments had sample data for their knowledge bases.

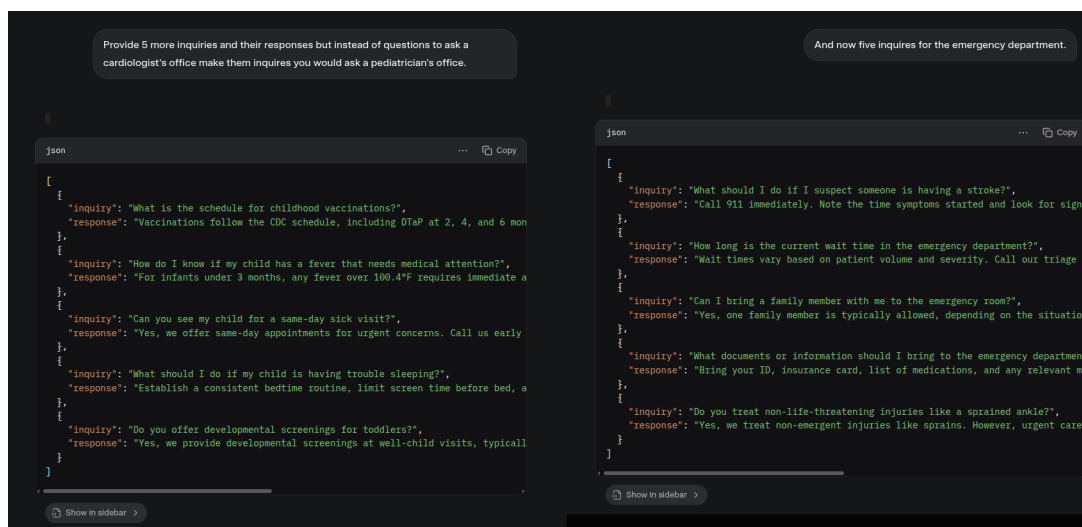


Figure 6: Grok was used to generate knowledge base entries in JSON format.

The knowledge base for a given department is loaded into context when the inquiry first arrives at that agent. This can be seen line 18 of cell 6 (see figure 4.) If the inquiry is coming from the operator the department-specific `SystemMessage` is injected into conversation history. This message is built dynamically using the `get_system_message_for_agent` utility function defined in cell 4. It uses other abstracted functions to load the proper JSON and inject it into the prompt. This is one of several ways to load documents into context. While this was simple to implement it has drawbacks. The department's knowledge base, in its entirety, is counting against our token limits (including the questions we are not using.) An improvement to this agent would be to introduce retrieval nodes that select the relevant questions from a document store in a separate step before generating the response. This would take weight off of our queries and conversation history.

Requirement 7: Trial runs

There are six prompts required for this assignment. All six run successfully and return expected responses (an answer based on a related knowledge base entry or a message stating no such knowledge base entry exists.) However, this does not showcase all of the features in these agents (notably, multi-turn conversations.) Because of the interactive nature of this product it is difficult to show all permutations of how to use the bot in a written report. The recorded video for this assignment contains many trials and various permutations of agent behavior. It is stored on my NU Google Drive and can be accessed here:

<https://drive.google.com/file/d/1IsNPPwMN7NijN8GKMiPsc1RXaHa7gqhH/view>

Cell 15 shows how the application loop is constructed. It is then utilized for one of the trial prompts. In cell 15's output we demonstrate a multi-turn conversation. Some of the inquiries match knowledge base context and some do not. We can also see the completion check working when the agent acknowledges the user is done. I will conclude with the transcript (figure 7) of one more trial, invoking several agent features.

```
User:
Hello?
Hello there, This is Northwestern Memorial Hospital, How can I assist you today?

User:
My child has a runny nose.
Pediatrics:: A runny nose can be a symptom of various conditions, including RSV
(Respiratory Syncytial Virus). If your child is experiencing additional symptoms
such as a cough, fever, or wheezing, it may be important to monitor them
closely. For infants, watch for severe signs like difficulty breathing, nasal
flaring, chest retractions, or bluish lips. If you have concerns or if your
child is in a high-risk group, please seek medical care. If you need further
assistance, feel free to ask!

User:
When does this become an emergency?
Pediatrics:: This situation becomes an emergency if your child shows severe
signs such as difficulty breathing, nasal flaring, chest retractions, or bluish
lips. Additionally, if your child is experiencing rapid breathing or
dehydration, you should seek medical care immediately. It's especially important
to monitor infants under 6 months or those with heart or lung issues. If you
have any concerns, please don't hesitate to reach out for help.

User:
How much would that visit cost?
Pediatrics:: I'm sorry, but I don't have information regarding the cost of
visits. I recommend contacting the hospital directly for details on pricing and
insurance coverage. If you have any other questions or need further assistance,
feel free to ask!

User:
I'm done. Thank you.
Pediatrics:: You're welcome! If you have any more questions in the future, feel
free to reach out. Have a great day!

User:
q
Goodbye!
```

Figure 7: An example session that invokes multiple agent features.