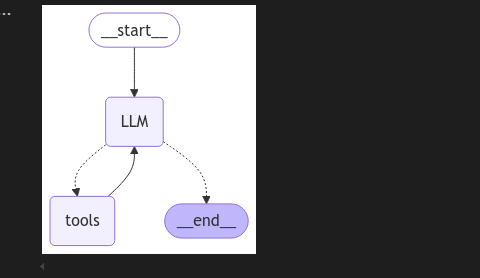
**Agent Design Document**

**1. Architecture Overview**

The agent uses LangGraph to create a stateful workflow. The main components are the LLM node and the tools node. The graph starts with the LLM processing the input, then conditionally routing to tools if needed, and looping back until no more tools are required.

The framework follows a LangGraph-based architecture with 3 key components:

1. Core Orchestrator: StateGraph manages workflow with 2 nodes:
   * LLM Node: Handles intent detection and general responses
   * Tools Node: Executes domain-specific functions
2. Memory System: ConversationBufferMemory persists chat history to JSON
3. Tool Layer: Modular functions for specific domains (weather, news, finance, translation and sentiment analysis)



**How this architecture is working :**

1. **Query Handling by LLM**

The workflow starts at the \_\_start\_\_ node.

The query is first handled by the LLM (Large Language Model) node.

1. **Intent Detection**

An intent detection function analyzes the query to determine its intent and extract relevant entities.

1. **Decision Making by LLM**

Based on the detected intent and entities, the LLM decides whether:

* 1. It can directly respond to the query (for general questions).
  2. A specific tool needs to be called to fetch or process additional information.

1. **Tool Invocation (if needed)**

If the LLM determines that a tool is required, it selects the appropriate tool.

The specified tool is then invoked to process the request and return the necessary information.

1. **Response Generation**

The output from the tool (if used) is sent back to the LLM.

The LLM generates the final response based on the tool's output.

1. **Workflow Completion**

The final response is presented to the user.

The workflow concludes at the \_\_end\_\_ node.

**2. Query Processing & Classification**

The detect\_intent function uses an LLM to classify the query's intent (weather, news, finance, sentiment, translation or general etc.) and extract entities (locations, companies). This classification determines which tool to use.

1. Intent Detection:
   * Uses Azure OpenAI's detect\_intent() with structured JSON output:

{

"intent": "weather/news/finance/sentiment/translation/general",

"entities": {

"locations": [],

"keywords": [],

"companies": []

}

* + }

1. Classification Logic:
   * Predefined intent categories via LLM prompt engineering

**3. Query-to-Function Mapping**

The route\_query function directs the query based on the detected intent. Each intent corresponds to a specific tool (e.g., weather intent triggers get\_weather).

Uses a dual mapping system:

1. Automatic Tool Selection (Primary):
   * LLM automatically chooses from pre-defined tools using bind\_tools()
   * Available tools: Weather, News, Finance, Translation, Sentiment
2. Explicit Routing (Fallback):

| **Intent** | **Function Called** | **Parameters Used** |
| --- | --- | --- |
| Weather | get\_weather() | First location from entities |
| News | get\_news() | Combined keywords |
| Finance | get\_financial\_data() | First company from entities |
| Translation | translate\_text() | Raw query text |
| Sentiment | analyze\_sentiment() | Raw query text |
| General | Direct LLM response | N/A |

**4. Key System Features**

Memory Management:

1. Stores conversations in JSON file
2. Checks for duplicate queries before processing
3. Uses ConversationBufferMemory to maintain chat history

Edge Case Handling:

1. Cached responses for repeated queries
2. Fallback to LLM for unclassifiable intents
3. Fallback to general chat for unrecognized queries

Multi-Query Support:

The current code routes to a single tool per query. For a query like "Stock price of Apple and sentiment around it", the agent would need to call `get\_financial\_data` and `analyze\_sentiment` sequentially. The existing setup can support handling multi-intent queries.

1. Stateful Messages State maintains conversation context
2. Sequential processing through LangGraph's cyclic workflow:

Input → LLM → Tool Check → Tool Execution → LLM → Final Output