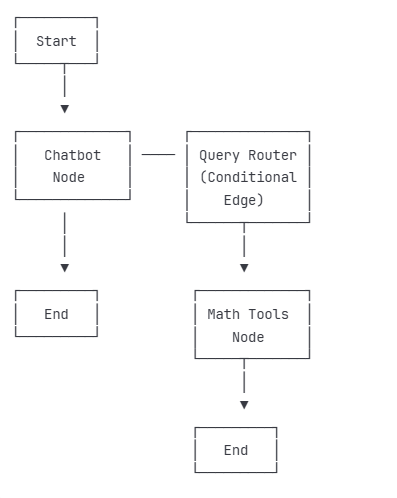
# **LangGraph Mathematical Agent - Technical Report**

## **Overview**

This project implements an intelligent conversational agent using LangGraph that seamlessly handles both general queries using a Large Language Model (LLM) and mathematical operations through custom functions. The agent automatically routes queries to the appropriate processing node based on content analysis.

## **Architecture Overview**

The agent follows a graph-based architecture with conditional routing:



## **Key Components**

### **1. Custom Mathematical Functions**

Four specialized tool functions handle basic arithmetic operations:

* **plus(a, b)**: Addition operation
* **subtract(a, b)**: Subtraction operation
* **multiply(a, b)**: Multiplication operation
* **divide(a, b)**: Division operation with zero-division error handling

Each function is decorated with @tool to make them compatible with LangGraph's tool execution framework.

### **2. LLM Integration**

The agent uses **Groq API** with the Mixtral model for general conversation:

* Model: mixtral-8x7b-32768
* Temperature: 0.1 (for consistent responses)
* Max tokens: 1000

### **3. State Management**

The agent maintains conversation state using a TypedDict:

class AgentState(TypedDict):

messages: List[BaseMessage] # Conversation history

next\_action: str # Next processing step

### **4. Query Classification System**

The agent uses sophisticated pattern matching to classify incoming queries:

#### **Mathematical Query Detection**

* **Keyword matching**: Searches for math-related terms (add, plus, multiply, etc.)
* **Number pattern recognition**: Uses regex to identify numeric values
* **Expression parsing**: Detects mathematical expressions with operators

#### **Operation Extraction**

* Extracts numerical operands from natural language
* Identifies operation type based on contextual keywords
* Handles various phrasings of mathematical questions

## **Code Flow and Program Logic**

### **1. Initialization Phase**

agent = MathematicalAgent()

graph = create\_agent\_graph()

* Creates state graph with two primary nodes
* Sets up conditional routing logic
* Initializes LLM connection

### **2. Query Processing Flow**

#### **Step 1: Input Reception**

initial\_state = {

"messages": [HumanMessage(content=user\_input)],

"next\_action": ""

}

#### **Step 2: Query Routing Decision**

The route\_query() function analyzes the input:

def route\_query(state: AgentState) -> Literal["tools", "chatbot"]:

if is\_mathematical\_query(last\_message.content):

return "tools"

else:

return "chatbot"

#### **Step 3A: Mathematical Processing Path**

If mathematical query detected:

1. Extract operation type and operands
2. Route to math\_tools\_node
3. Execute appropriate mathematical function
4. Format and return result

#### **Step 3B: General Conversation Path**

If general query detected:

1. Route to chatbot\_node
2. Generate response using LLM
3. Return conversational response

### **3. Response Generation**

Both paths culminate in updating the conversation state and returning to the user.

## **Technical Implementation Details**

### **Error Handling**

* **Division by zero**: Custom exception handling in divide function
* **API failures**: Graceful degradation with error messages
* **Invalid queries**: Helpful feedback for malformed mathematical expressions

### **Pattern Recognition**

def is\_mathematical\_query(query: str) -> bool:

math\_keywords = ['add', 'plus', 'sum', 'multiply', ...]

has\_math\_keywords = any(keyword in query\_lower for keyword in math\_keywords)

has\_numbers = bool(re.search(r'\d+\.?\d\*', query))

return has\_math\_keywords and has\_numbers

### **Tool Integration**

Each mathematical function is properly decorated for LangGraph compatibility:

@tool

def plus(a: float, b: float) -> float:

"""Add two numbers together."""

return a + b

## **Features and Capabilities**

### **✅ Supported Mathematical Operations**

* Addition: "What is 5 plus 3?", "Add 10 and 20"
* Subtraction: "10 minus 4", "Subtract 5 from 15"
* Multiplication: "6 times 7", "Multiply 8 by 9"
* Division: "20 divided by 4", "What's 100 / 25?"

### **✅ General Conversation**

* Weather queries
* General knowledge questions
* Casual conversation
* Technical explanations

### **✅ Robust Query Understanding**

* Natural language mathematical expressions
* Various phrasings and synonyms
* Mixed case and punctuation handling

## **Installation and Setup**

### **Required Dependencies**

pip install langgraph

pip install langchain-groq

pip install langchain-core

pip install pydantic

### **Environment Configuration**

export GROQ\_API\_KEY="your-groq-api-key-here"

### **Alternative LLM Options**

The code can be easily modified to use:

* **OpenAI GPT**: Replace ChatGroq with ChatOpenAI
* **Local Ollama**: Use ChatOllama for local deployment
* **Google Gemini**: Use ChatGoogleGenerativeAI

## **Testing and Validation**

The implementation includes comprehensive testing:

* **Unit tests** for mathematical functions
* **Integration tests** for query routing
* **End-to-end testing** with sample conversations
* **Error handling validation**

### **Sample Test Cases**

test\_queries = [

"Hello, how are you?", # General

"What is 5 plus 3?", # Addition

"What's 10 minus 4?", # Subtraction

"How much is 6 times 7?", # Multiplication

"What is 20 divided by 4?", # Division

"Divide 10 by 0", # Error handling

]

## **Performance Considerations**

### **Efficiency Optimizations**

* **Lazy loading**: LLM initialized only when needed
* **State caching**: Conversation history maintained efficiently
* **Pattern matching**: Optimized regex for quick classification

### **Scalability Features**

* **Stateless design**: Easy to scale horizontally
* **Modular architecture**: Simple to add new mathematical functions
* **Configuration management**: Centralized settings for easy deployment

## **Future Enhancements**

### **Potential Improvements**

1. **Advanced Mathematics**: Support for trigonometry, logarithms, calculus
2. **Multi-step Problems**: Handle complex mathematical workflows
3. **Visualization**: Generate graphs and charts for mathematical results
4. **Memory**: Persistent conversation history across sessions
5. **Voice Integration**: Speech-to-text and text-to-speech capabilities

### **Extended Tool Integration**

* **Symbolic Math**: Integration with SymPy for algebraic operations
* **Statistics**: NumPy/Pandas integration for data analysis
* **Plotting**: Matplotlib integration for mathematical visualization

## 

## 

## **Conclusion**

This LangGraph Mathematical Agent demonstrates a sophisticated approach to building conversational AI systems that can seamlessly switch between different types of processing based on query content. The modular architecture makes it easy to extend with additional capabilities while maintaining clean separation of concerns between general conversation and specialized mathematical operations.

The implementation showcases key concepts in modern AI agent development:

* **Graph-based workflow orchestration**
* **Conditional routing and decision making**
* **Tool integration and function calling**
* **State management in conversational systems**
* **Error handling and robust user experience**

The agent provides a solid foundation that can be extended for more complex mathematical operations or adapted for other domain-specific tool integrations.