# **Agentic AI: A Practical Tutorial**

*Author:TB | Version: 1.0 | Date: 05/05/2025 | Initial creation*

## **1. Introduction to Agentic AI**

**Definition**: Autonomous systems that:

* **Perceive**: Understand user inputs (text/voice/etc.)
* **Reason**: Decide actions using LLMs + rules
* **Act**: Execute tools (APIs, calculations, etc.)
* **Learn** (optional): Improve through feedback

**Key Characteristics**:

* Tool-oriented (vs. pure chat)
* Fail-safe design (fallback mechanisms)
* Context-aware execution

## **2. Core Architecture**

### **Agent Class Overview**

The Agent class orchestrates:

1. **Tool Registry**: Maintains available functions
2. **LLM Router**: Selects appropriate tools
3. **Fallback System**: Handles LLM failures

*Key Design Choices*:

* Strict input/output formats for tools
* Layered decision-making (LLM → Rules)
* Isolated error handling

## **4. Implementation Deep Dive**

### **Tool Definition**

@tool

def search(query: str) -> str:

"""Standardized search interface"""

*# ... (mock implementation)*

(Please refer <https://www.freecodecamp.org/news/python-decorators-explained/> to understand python decorators )

### **Execution Flow**

1. User calls agent.run("Find cats")
2. System:
   * Classifies task → search
   * Validates input
   * Executes tool
   * Returns formatted result

### **Critical Components**

#### **LLM Prompting**

llm\_prompt = f"""Classify the task:

Options: search|add\_numbers|none

Task: "{prompt}"

Response: """

#### **Fallback Mechanism**

if not llm\_result:

return self.\_keyword\_match(prompt) *# Regex-based*

## **5. Hands-On Exercises**

### **Exercise 1: Extend Toolset**

**Task**: Add a weather tool that:

* Matches prompts like "weather in Kolkata"
* Returns mocked data

**Steps**:

1. Add new tool function
2. Extend keyword fallback
3. Test with:
4. print(agent.run("Weather forecast"))

### **Exercise 2: Improve Error Handling**

**Problem**:

print(agent.run("Add")) *# Poor error message*

**Solution**:  
Modify add\_numbers() to:

if not re.search(r"\d+", prompt):

return "Error: No numbers detected"

### **6. Debugging Guide**

### **Common Issues and How to Resolve Them**

| **Symptom** | **Debugging Steps** | **Solution** |
| --- | --- | --- |
| **LLM selects wrong tool** | **Add print(f"LLM Output: '{response}'") in \_try\_llm\_selection()** | **Tighten prompt instructions (e.g., "Respond ONLY with 'search' or 'add\_numbers'"). Test with edge cases like "Tell a joke".** |
| **Math tool false positives** | **Add print(f"Extracted numbers: {re.findall(r'\d+', prompt)}") in add\_numbers()** | **Strengthen input validation: Reject prompts with no numbers (e.g., "Add" without digits).** |
| **LLM times out/hangs** | **Check GPU memory usage (nvidia-smi). Monitor max\_new\_tokens (current: 5).** | **Reduce max\_new\_tokens further (e.g., 3) or switch to a smaller model (flan-t5-base).** |
| **Keyword fallback fails** | **Add print(f"Matched keywords: {re.search(pattern, prompt)}") in \_try\_keyword\_fallback()** | **Expand regex patterns (e.g., add \b** |

### 

### **Debug Template**

*# Add to \_try\_llm\_selection():*

print(f"LLM Input:\n{llm\_prompt}\nRaw Output: '{response}'")

## **7. Best Practices**

### **Tool Development**

1. **Single Responsibility**: Each tool does one thing
2. **Input Validation**: Validate before processing
3. **Clear Errors**: Help users correct mistakes

### **LLM Optimization**

* Start with small prompts
* Gradually add examples if needed
* Test with edge cases

**Appendix**:

* Complete Code: Available separately
* Hugging Face Docs: [huggingface.co/docs](https://huggingface.co/docs)