



KDD LAB **Agentic AI** Internship Task

Project Title:

Task-Oriented AI Agent Using LangChain and Tool Integration

Objective:

Build a functional **agentic AI system** using **LangChain** and a foundational **language model (LLM)** that can solve user-defined tasks by utilizing memory, tools (e.g., search, calculator), and reasoning. This task will help you explore **agent design, multi-step reasoning, and decision chaining**, which are core to agentic AI systems like personal assistants, coding agents, and planning bots.

Task Breakdown

Part 1: Environment Setup and Tool Familiarization

- Install and set up:
 - Python (3.9+)
 - Core libraries: **langchain**, **openai** or **ollama**, **faiss**, **pandas**, **requests**
 - Optional tools: **ChromaDB**, **DuckDuckGoSearch**, **PythonREPLTool**, **SerpAPI**, etc.
- Explore **LangChain's agent types**: Zero-shot, **ReAct**, and Conversational agents.
- Create a basic agent that can answer simple questions using **OpenAI** or a local LLM via **Ollama** or **Hugging Face Transformers**.

Part 2: Build a Functional Agent with Tools

- Define a **task-oriented agent** (e.g., research assistant, report summarizer, personal finance advisor).
- Integrate at least **3 tools** with the agent:
 - A **search tool** (e.g., **DuckDuckGoSearch** or SerpAPI)
 - A **Python REPL tool**
 - A **memory component** (e.g., **conversational memory** or vector store)
- Define a **prompt template** for guiding the agent's reasoning.
- Allow multi-step reasoning — the **agent should autonomously decide which tools to use and in what order.**

Part 3: User Interaction Interface (Optional Task)

- **CLI** or Streamlit/Flask interface:
 - Allow users to **input a query** (e.g., "Summarize latest news on AI regulation and give me a cost estimate of deploying a model").
 - Show each reasoning step and the final result.
- **Display:**
 - Intermediate decisions
 - Final outcome
 - Tool usage log

Part 4: Evaluation & Analysis

- Test your agent on **5+ real-world queries** or use cases (e.g., trip planning, investing assistant, resume enhancements).
- Evaluate:
 - Tool selection efficiency
 - Relevance of outputs
 - Error handling when the agent fails or gets stuck
- Analyze:
 - Agent's behavior across prompt variations
 - Memory effectiveness
 - Limitations of current LangChain-based systems

Bonus Part (Completely Optional)

- Add **long-term memory** with a vector database (e.g., Chroma or FAISS).
- Use **LangGraph** to model and visualize the agent's workflow graphically.
- Extend your agent to **trigger APIs or execute commands** (with safeguards).

Deliverables

- Python **script** or Jupyter notebook **implementing the agent**.
- If a web UI is built, provide **deployment instructions** (localhost or GitHub).
- Short **video walkthrough (Demo)** of your system:
 - Agent **architecture & components**
 - **Tool integration**
 - **Query examples and live demonstration**
- Include a **README file** with:
 - **Setup steps**
 - **Tool choices and rationale**
 - **Known issues** (if any arise) and next steps
- **GitHub repo encouraged to be submitted** alongside.

Disclaimer & Development Integrity Guidelines

This task is meant to test your core understanding of **agent-based system design** and the LangChain ecosystem. Please adhere to the following:

- Do **not use GPT, Claude, Gemini, Copilot**, or similar tools to generate the full code or entire architecture without core understanding.
- You **may consult tutorials or documentation**, but the final code and system logic must be your own.
- You will be required to **explain each part of your implementation**, especially the **reasoning engine, prompt design, and tool integrations**.
- Submissions that include unexplained, AI-generated, or copied code will be **disqualified**.

Key Libraries & Frameworks You May Use

- **Python** – Core development language
- **LangChain** – Agent framework for tool use, chaining, memory
- **PyTorch / TensorFlow** – (Optional) if building or fine-tuning models
- **scikit-learn / pandas / NumPy** – Data processing
- **FAISS / Chroma** – Vector stores for memory
- **Streamlit / Flask** – UI development