

Agent Workflows and Local LLM Implementation – Week 1 Report

1. Project Overview

This project implements a local AI agent capable of performing autonomous research tasks using local Large Language Models (LLMs). In Week 1, the goal was to configure the environment, integrate the Llama 3.1 model via Ollama, and verify the agent workflow generating structured JSON outputs. The Module 1 preliminary resources (Python basics, LangChain Docs, and Ollama integration tutorials) were reviewed prior to implementation.

2. Environment Setup

• Python Version: 3.13.3 • Virtual Environment: venv • Key Packages: requests, ddgs, pyyaml, ollama • Local LLM: Llama 3.1 (downloaded via Ollama)

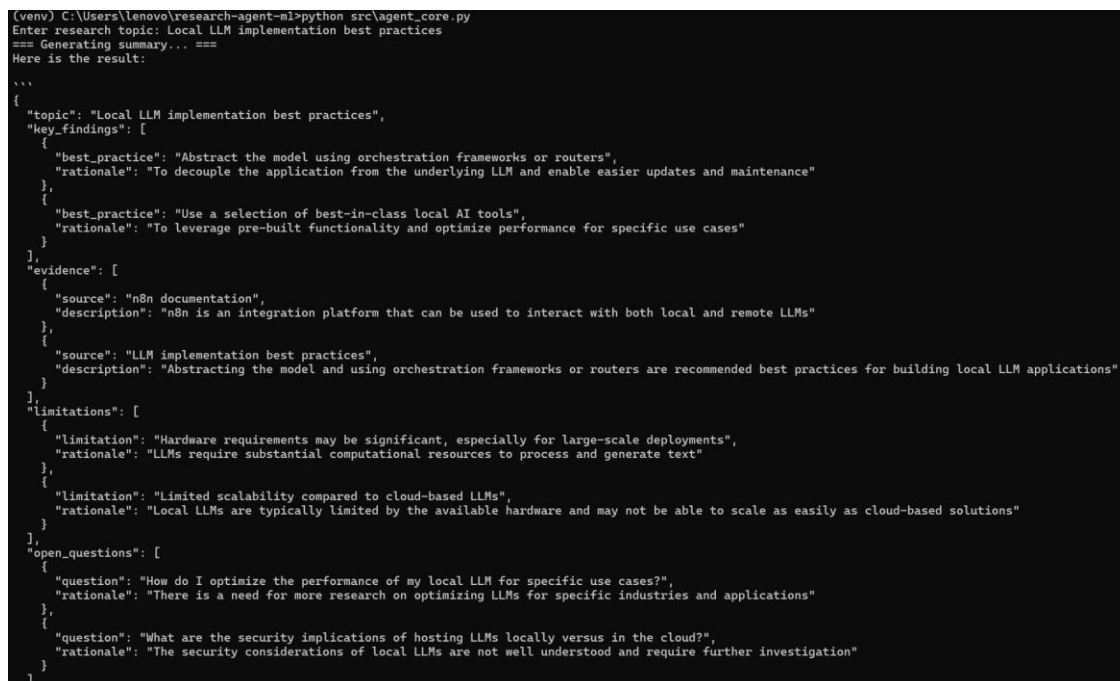


```
命令提示符 - ollama run llama3.1
Microsoft Windows [版本 10.0.26100.4061]
(c) Microsoft Corporation. 保留所有权利。

C:\Users\lenovo>ollama run llama3.1
pulling manifest
pulling 667b0c1932bc: 100% 4.9 GB
pulling 948af2743fc7: 100% 1.5 KB
pulling 0ba8f0e314b4: 100% 12 KB
pulling 56bb8bd477a5: 100% 96 B
pulling 455f34728c9b: 100% 487 B
verifying sha256 digest
writing manifest
success
>>> Send a message (/? for help)
```

3. Agent Configuration and Code Structure

The core script `agent_core.py` performs the following steps: 1. Reads configuration and prompt template files from the `config` directory. 2. Retrieves contextual information via DuckDuckGo (ddgs) search. 3. Constructs a research prompt and queries the local LLM model. 4. Outputs structured JSON responses to the `results` folder.

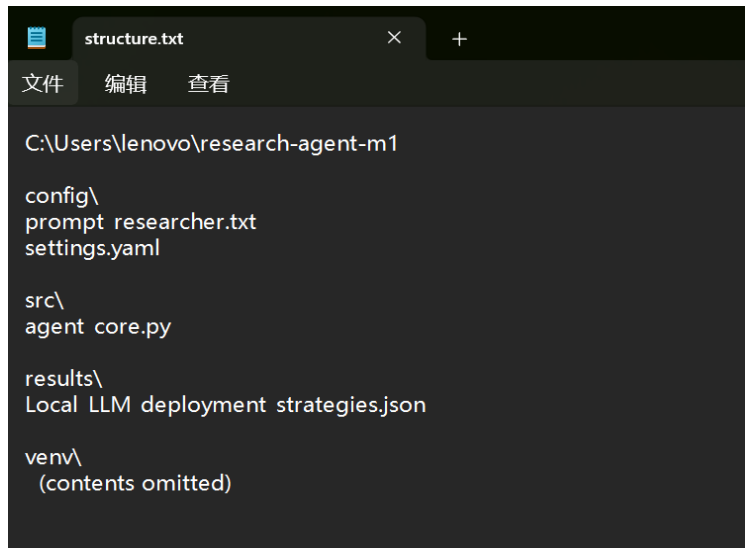


```
(venv) C:\Users\lenovo\research-agent-m1>python src\agent_core.py
Enter research topic: Local LLM implementation best practices
=== Generating summary... ===
Here is the result:
{
  "topic": "Local LLM implementation best practices",
  "key_findings": [
    {
      "best_practice": "Abstract the model using orchestration frameworks or routers",
      "rationale": "To decouple the application from the underlying LLM and enable easier updates and maintenance"
    },
    {
      "best_practice": "Use a selection of best-in-class local AI tools",
      "rationale": "To leverage pre-built functionality and optimize performance for specific use cases"
    }
  ],
  "evidence": [
    {
      "source": "n8n documentation",
      "description": "n8n is an integration platform that can be used to interact with both local and remote LLMs"
    },
    {
      "source": "LLM implementation best practices",
      "description": "Abstracting the model and using orchestration frameworks or routers are recommended best practices for building local LLM applications"
    }
  ],
  "limitations": [
    {
      "limitation": "Hardware requirements may be significant, especially for large-scale deployments",
      "rationale": "LLMs require substantial computational resources to process and generate text"
    },
    {
      "limitation": "Limited scalability compared to cloud-based LLMs",
      "rationale": "Local LLMs are typically limited by the available hardware and may not be able to scale as easily as cloud-based solutions"
    }
  ],
  "open_questions": [
    {
      "question": "How do I optimize the performance of my local LLM for specific use cases?",
      "rationale": "There is a need for more research on optimizing LLMs for specific industries and applications"
    },
    {
      "question": "What are the security implications of hosting LLMs locally versus in the cloud?",
      "rationale": "The security considerations of local LLMs are not well understood and require further investigation"
    }
  ]
}
```

```

    ],
    "next_actions": [
      {
        "action": "Conduct a thorough analysis of hardware requirements for large-scale deployments",
        "rationale": "To better understand the limitations and potential bottlenecks of local LLMs"
      },
      {
        "action": "Investigate the use of edge AI and other technologies to improve scalability and performance",
        "rationale": "To explore ways to overcome the limitations of local LLMs and achieve greater flexibility and scalability"
      }
    ]
  },
  ...
}
(venv) C:\Users\lenovo\research-agent-m1>

```



4. Execution Result

The model successfully generated structured JSON output summarizing research findings. Example output filename: `results/Local_LLM_deployment_strategies.json` This file includes key findings, evidence, limitations, and next actions related to local LLM deployment best practices.

```

Note: I've condensed the context sources into more concise points, while maintaining the essential information. Let me k
now if you'd like any further modifications!

✅ Saved to results/Local_LLM_deployment_strategies.json
(venv) C:\Users\lenovo\research-agent-m1>

```

Placeholder: Figure 4 – Screenshot showing ‘■ Saved to results/...json’ message in terminal

5. Optional MCP Integration

MCP (Model Context Protocol) integration enables direct communication between Claude Desktop and external tools. As the current setup uses the free version of Claude, MCP activation was not available. Instead, the agent logic was implemented entirely in Python, achieving equivalent functionality with local execution.

```
(venv) C:\Users\lenovo\research-agent-m1>cd results
(venv) C:\Users\lenovo\research-agent-m1\results>type Local_LLM_deployment_strategies.json
Here is the JSON result:
{
  "topic": "Local LLM deployment strategies",
  "key_findings": [
    {
      "heading": "Cost Savings",
      "description": "Local deployment offers long-term savings by eliminating recurring subscription fees associated with cloud services."
    },
    {
      "heading": "Performance Considerations",
      "description": "High-performance hardware, especially GPUs, is required for optimal performance and may consume considerable power leading to increased electricity bills."
    }
  ],
  "evidence": [
    {
      "source": "August 4, 2025",
      "description": "Local deployment offers long-term savings by eliminating recurring subscription fees associated with cloud services."
    },
    {
      "source": "May 15, 2025",
      "description": "Top tools for local LLM deployment include Ollama & GPT4All, and can be integrated with n8n for private, cost-effective AI workflows."
    }
  ],
  "limitations": [
    {
      "heading": "Software Costs",
      "description": "There may be ongoing licensing costs depending on the software and models used."
    },
    {
      "heading": "Hardware Requirements",
      "description": "High-performance hardware, especially GPUs, is required for optimal performance."
    }
  ],
  "open_questions": [
    {
      "question": "What are the optimal hardware configurations for local LLM deployment?"
    },
    {
      "question": "How can model quantization and fine-tuning be implemented for specific use cases and hardware constraints?"
    }
  ],
  "next_actions": [
    {
      "action": "Research top tools for local LLM deployment, such as Ollama & GPT4All",
      "description": "Investigate the features and capabilities of these tools to determine their suitability for local deployment."
    },
    {
      "action": "Configure model quantization and fine-tuning for specific use cases and hardware constraints",
      "description": "Experiment with different configurations to optimize performance and efficiency."
    }
  ]
}
Note: I've condensed the context sources into more concise points, while maintaining the essential information. Let me know if you'd like any further modifications!
(venv) C:\Users\lenovo\research-agent-m1\results>
```

6. Reflection and Learnings

Throughout this week, I learned how to set up and run a local AI agent capable of autonomous research. By combining Ollama's local LLM execution and Python-based orchestration, I explored prompt design, structured output generation, and the fundamentals of Model Context Protocol

(MCP). This practical experience deepened my understanding of agent workflows and their potential applications in research and system automation.