Day97_Practical_Agentic_AI_with_AGNO

October 14, 2025

Practical: Agentic AI with AGNO

1 Introduction

What we learned yesterday (Day 96 theory recap):

- **Agentic AI** refers to AI systems that act autonomously to achieve goals. Unlike simple AI, it can **plan, reason, and take sequential actions**.

2 Key Concepts:

- **Agent:** An AI entity that perceives its environment and acts.
- **Agency:** The ability to act independently and make decisions.
- **Tools:** External functions or modules the agent can use (e.g., calculator, web search).
- **Reasoning:** Deductive, inductive, or abductive logic used to achieve goals.

3 Types of Agents:

1. **Simple Agent:** Executes tasks directly.
2. **Reasoning Agent:** Plans multi-step actions using tools.
3. **Multimodal Agent:** Handles different input/output types (text, image, audio).

4 Frameworks & APIs for Agentic AI:

4.1 Agent Frameworks:

- **AGNO (formerly PhiData)** — lightweight, reliable, supports multi-agent and multi-tool setups, convenient without Google services.
- **LangChain** — Python framework for building agents, chains, and integrations with LLMs.
- **AutoGPT / BabyAGI** — autonomous agents for goal-driven tasks using LLMs.
- **Hugging Face Transformers** — wide range of pre-trained models for NLP, vision, and multimodal tasks.
- **LlamaIndex (GPT Index)** — tool to structure knowledge for agents.
- **OpenAI Function Calling** — enables LLMs to call external tools via API.

4.2 LLM Options:

Type	Name / Framework	Access / Cost	Notes
Free Cloud	OpenAI GPT-3.5	Free tier available	Great for prototyping, limited tokens/min
	OpenAI GPT-4	Paid	Better reasoning, higher token limit
	Cohere	Free & Paid	NLP-focused, embeddings available
	Anthropic Claude	Free tier & Paid	Strong alignment and reasoning
Paid Cloud	OpenAI GPT-4 / GPT-3.5 Turbo	Paid	Industry standard, cloud API
	Groq LLM (llama-3.1-8b-instant)	Paid	High-speed, supports multi-agent streaming
	AI21 Studio (Jurassic-2)	Paid	Text generation & reasoning
Local / Open Source	LLaMA (Meta)	Free	Requires GPU, good for experimentation
	MPT (MosaicML)	Free & Paid	Open weights, fast training options
	Falcon 7B / 40B	Free	Open weights, good for reasoning
	GPT4All	Free	Local chat model, easy setup
	Vicuna / Alpaca	Free	Fine-tuned LLaMA derivatives, lightweight
Hybrid / Embeddings	Sentence-Transformers	Free	Generate vector embeddings for retrieval agents
	ChromaDB / Weaviate	Free & Paid	Vector DB for knowledge agents

Tip: Start with **AGNO + a free cloud LLM** for learning, then move to **paid or local LLMs** for more intensive projects.

5 Practical Goal for Today:

- Implement:
  1. **Simple Agent**
  2. **Reasoning Agent with Tools**
  3. **Multi-Tool Agent**
  4. **Multi-Agent Team**

  We will use **AGNO** to create agents, integrate tools (calculator, translator, web search, finance data), and run multi-agent teamwork to solve complex tasks.

5.1 Why AGNO:

- Reliable and lightweight.
- Supports **tool integration** and **multi-agent coordination**.
- Convenient for offline testing.
- Almost all features work out-of-the-box except some integrations (e.g., Google services).

5.2 Hands-On Exercises / Possible Workflows:

- Create a **single agent** that answers questions.
- Create an agent using **tools** (calculator, translator).
- Build a **multi-tool agent** that can solve different types of tasks.
- Build a **multi-agent team** (Web Agent + Finance Agent) and coordinate tasks.
- Extend agents with **custom tools** for finance, web scraping, translation, or table summarization.

  Note:
  This notebook is used primarily for **documentation and practical demonstration** of Agentic AI with AGNO.

  The output formatting in the notebook is designed for **interactive display**, which may appear differently when exported to PDF.

  Streamed or rich outputs (like agent responses) are **not fully preserved in PDF exports**.

  To see the actual results, always **run the notebook directly**.

6 Simple Agent

This agent answers a simple question without using any tools.

- We imported Agent and Groq from AGNO.
- agent1 is a simple AI agent that uses the Groq model.
- print_response sends a prompt to the agent and prints the output.

[14]:
```python
from agno.agent import Agent
from agno.models.groq import Groq
from dotenv import load_dotenv

# Load environment variables (API keys)
load_dotenv()

# Create a simple AI agent
agent1 = Agent(
    model=Groq(id="llama-3.1-8b-instant"),  # Groq model used
    description="You are a smart assistant that answers clearly and simply.",
    markdown=True
)

# Ask the agent a question
agent1.print_response("Hello! What is Agentic AI?", stream=True)
```

```
Output()
```

Note: For proper formatting, view this output in the notebook; PDF export may not preserve it.

7 Reasoning Agent with One Tool (Calculator)

Here, we add a Calculator tool for the agent.

- Tools are passed as a dictionary: `{ "ToolName": function }`.
- The agent can now call the calculator tool when prompted.

[15]:
```
# Define the calculator function
def calculator(expression):
    try:
        return str(eval(expression))  # Evaluate math expression
    except Exception as e:
        return f"Error: {e}"

# Create agent and pass tools as a dictionary
agent2 = Agent(
    model=Groq(id="llama-3.1-8b-instant"),
    description="You are an agent that can use tools when needed.",
    tools={"Calculator": calculator},  # Add the calculator tool
    markdown=True
)

# Test the Calculator tool
agent2.print_response("Use the Calculator to compute 25 * 4 + 10", stream=True)
```

```
Output()
```

Note: For proper formatting, view this output in the notebook; PDF export may not preserve it.

8 Agent with Two Tools (Calculator + Translator)

Now the agent can do math calculations and translate text.

- Multiple tools can be passed in the tools dictionary.
- The agent can decide which tool to use based on the prompt.

[16]:
```
# Tool 1: Calculator
def calculator(expression):
    try:
        return str(eval(expression))
    except Exception as e:
        return f"Error: {e}"

# Tool 2: Translator
def translator(text):
    translations = {"hello":"  ","good morning":"   "}
    return translations.get(text.lower(), "Translation not found")
```

```
# Create agent with both tools
agent3 = Agent(
    model=Groq(id="llama-3.1-8b-instant"),
    description="Agent can calculate and translate.",
    tools={
        "Calculator": calculator,
        "Translator": translator
    },
    markdown=True
)

# Test both tools
agent3.print_response("Translate 'hello' using Translator", stream=True)
agent3.print_response("Use Calculator to solve 50*5+20", stream=True)
```

```
Output()
```

```
Output()
```

Note: For proper formatting, view this output in the notebook; PDF export may not preserve it.

9 Multi-Agent Setup (Web + Finance)

Here we create two specialized agents and a coordinator function to handle multi-agent tasks.

- `web_agent` searches the web using DuckDuckGo.
- `finance_agent` fetches financial data using YFinance.
- `team_task()` coordinates both agents manually (latest AGNO doesn't support `team=[...]`).
- This allows **multi-agent + multi-tool execution** in one go.

[19]:
```
# Multi-Agent AGNO Example



# 1 Import required modules
from agno.agent import Agent
from agno.models.groq import Groq
from agno.tools.duckduckgo import DuckDuckGoTools
from agno.tools.yfinance import YFinanceTools
from dotenv import load_dotenv

# Load environment variables (like API keys)
load_dotenv()

# 2 Create Web Agent
```

```
web_agent = Agent(
    name="Web Agent",
    role="Search the web for relevant information",
    model=Groq(id="llama-3.1-8b-instant"),
    tools=[DuckDuckGoTools()],
    instructions="Always include sources in your answer",
    markdown=True
)

# 3 Create Finance Agent
# Initialize YFinanceTools without arguments (parameters are used in method␣
 ↪calls)
finance_tools = YFinanceTools()

# Optional wrapper function to summarize stock data for brevity
def get_stock_summary(symbol):
    data = finance_tools.get_historical_stock_prices(
        symbol=symbol,
        period="1y",
        interval="1mo"
    )
    # Return only first 10 rows to avoid huge token usage
    return f"{symbol} stock summary (last 10 entries):\n{data[:10]}"

# Finance agent with a simplified tool
finance_agent = Agent(
    name="Finance Agent",
    role="Get financial data",
    model=Groq(id="llama-3.1-8b-instant"),
    tools={"StockSummary": get_stock_summary},  # Add wrapper function as tool
    instructions="Provide a brief summary in tables",
    markdown=True
)

# 4 Coordinator Logic (Manual Team Task)
def team_task(query):
    print(">>> Web Agent Response:")
    web_agent.print_response(query, stream=True)

    print("\n>>> Finance Agent Response:")
    # Example symbols; can be extended
    symbols = ["NVDA", "AMAT"]
    for symbol in symbols:
        summary = get_stock_summary(symbol)
        print(f"\n{summary}\n")

# 5 Run the Team Task
```

```
task = "What's the market outlook and financial performance of AI semiconductor␣
  ↪companies?"
team_task(task)
```

```
>>> Web Agent Response:

Output()



>>> Finance Agent Response:

NVDA stock summary (last 10 entries):
{"17304336


AMAT stock summary (last 10 entries):
{"17304336
```

Note: For proper formatting, view this output in the notebook; PDF export may not preserve it.

10 Summary / Key Notes

- **Step 1:** Simple agent → answers questions directly.
- **Step 2:** Added **one tool** (Calculator) for reasoning.
- **Step 3:** Added **two tools** (Calculator + Translator) → multi-tool agent.
- **Step 4: Multi-agent setup** → Web + Finance agents coordinated manually.
- AGNO is **fast, lightweight, and reliable,** unlike some services that require Google APIs or other dependencies.
- Using AGNO, you can **extend AI agents easily** with tools and multi-agent workflows.

Possible Practical Extensions

- Add **more tools** (Summarizer, WolframAlpha, Image Recognition).
- Create **multi-modal agents** (text + image).
- Build **fully autonomous agents** with sequential reasoning.
- Combine **team of agents** for research, finance, or data analysis workflows.