AI AGENTS AND AGENTIC AI WITH PYTHON & GENERATIVE AI COURSE

Module 1 _ Lecture Notes

INSTRUCTOR: DR. JULES WHITE
WRITTEN BY: HOUSHYAR JAFARI ASL

Abstract

Course: *AI Agents and Agentic AI with Python & Generative AI* Module 1 (Presented by Dr. Jules White)

This module introduces foundational concepts for building autonomous AI agents, focusing on **flipped interaction patterns**, **agent loops**, and **structured outputs**. Key topics include:

- **Flipped Interaction Paradigm**: Reversing traditional AI-user dynamics to enable agent-driven decision-making.
- **The Agent Loop Architecture**: A cyclical framework for autonomous action, feedback, and adaptation.
- **Programmatic Prompting**: Techniques to enforce structured, parseable outputs from LLMs (with CPU-optimized implementations).
- **Agent Memory & Environment Interfaces**: Methods for persistence and system integration.

The notes include **practical templates**, **case studies** (e.g., customer service agents), and **GPU-free code samples** (accessible via GitHub) designed for Google Colab. These resources emphasize **production-ready practices** while lowering barriers to entry by eliminating API/GPU dependencies.

Developed as a companion to Dr. Jules White's course, with added optimizations for broader accessibility.

Introduction

Link of the Course

Link of the Colab Notebook (Github)

What is Agentic AI?

Agentic AI refers to artificial intelligence systems that can **act with agency**—meaning they can:

- **Autonomously problem-solve**
- **React to different situations**
- **Take actions, observe outcomes, and adapt**

You may have already interacted with Agentic AI in generative AI systems without realizing it. The key difference is that Agentic AI is designed to **interact with the real world**, not just generate text or images.

Key Characteristics of Agentic AI:

1. **Autonomy** – Can make decisions independently.
2. **Adaptability** – Responds dynamically to changes or unexpected inputs.
3. **Task-Oriented** – Breaks down goals into actionable steps.
4. **Feedback-Driven** – Adjusts behavior based on results.

Example: Cooking with an Agentic AI System

To understand Agentic AI, let's simulate a **human-AI collaboration** where:

- The **AI provides step-by-step instructions** (agency).
- The **human carries out the actions** (execution).

Step 1: Defining the Prompt

The AI is given a structured prompt:

"You will help me cook. Since you can't physically act, you will give me instructions, and I will perform them. We will go one step at a time. Ask me what I want to cook."

Step 2: AI Takes Initiative

- The AI asks: *"What would you like to cook today?"*
- User responds with a complex request:

"A dish combining Ethiopian and Uzbek flavors, keto-friendly, under 1000 calories."

Step 3: AI Plans & Delegates Tasks

The AI:

1. **Generates a recipe** (marinated meat + spiced vegetables).
2. **Lists ingredients**.
3. **Gives step-by-step instructions**:
    - *"Marinate the meat with these spices… Let me know when done."*

Step 4: Human Feedback Loop

- User reports: *"Meat is marinated."*
- AI responds: *"Great! Now cook the meat (steps…). Let me know when done."*

Handling Unexpected Changes (Adaptability)

- **Problem:** User says: *"No vegetables, but I have durian fruit."*
- **AI Adapts:**
    - *"Durian is different, but let's adjust the recipe."*
    - Provides new instructions for durian preparation.

Key Observations:

- The AI **controls the workflow** (agency).
- It **adapts to real-time feedback** (burning meat, missing ingredients).
- The human **executes actions**, but the AI **directs the process**.

Why is Agentic AI Different?

Traditional software is **rigid**—if input doesn't match expectations, it fails.

Agentic AI is:
Flexible – Adjusts to unexpected inputs (e.g., durian instead of vegetables).
Interactive – Works in a feedback loop (human or system provides updates).
Goal-Oriented – Persists until the task is completed.

Challenges in Agentic AI

1. **Human-in-the-Loop vs. Full Automation**
    - Humans can interpret vague instructions, but machines need **structured inputs**.
    - Example: If AI says *"Yo, create this recipe,"* a spreadsheet won't understand.
2. **Bridging Generative AI with Rigid Systems**
    - Generative AI is **expressive** (explains, comments, adapts).
    - Traditional software expects **exact commands**.
    - **Solution:** Prompt engineering to ensure AI gives **machine-readable instructions**.

Key Takeaways

1. **Agentic AI = AI with autonomy & adaptability.**
2. **Works via iterative feedback loops (plan → act → observe → adjust).**
3. **Critical for real-world applications where conditions change.**
4. **Next step: Automating interactions (removing the human from the loop).**

Flipped Interaction Pattern

Introduction to Flipped Interaction

Agentic AI systems rely on a **"flipped interaction" pattern**, where:

- Instead of the **user directing the AI**, the **AI directs the user** (or other systems).
- The AI **asks questions** or **provides step-by-step instructions** to achieve a goal.

Core Concept

- Traditional AI: User asks questions → AI responds.
- **Agentic AI (Flipped Interaction):**
    - AI **takes initiative** by asking questions or giving tasks.
    - User (or another system) **provides feedback**.
    - AI **adapts dynamically** based on responses.

Key Components of the Flipped Interaction Pattern

1. **Step-by-Step Execution**
    - AI breaks down tasks into **sequential steps** (one at a time).
    - Example Prompt:

"Ask me questions one at a time to gather enough information to suggest a restaurant in Nashville tonight. Ask the first question."

2. **Adaptive Questioning**
    - AI **chooses the next question** based on previous answers.
    - Example:
        - AI: *"What type of cuisine do you want?"*
        - User: *"Tex-Mex."*
        - AI adapts: *"Any dietary restrictions?"*
3. **Handling Unstructured Input**
    - Users can respond **freely** (no rigid format required).
    - Example:
        - AI: *"Casual or upscale dining?"*
        - User: *"Kid-friendly."*
        - AI interprets and adjusts.

4. **Feedback-Driven Adaptation**
    - AI refines its approach based on real-time feedback.
    - Example:
        - AI suggests a restaurant 13 minutes away.
        - User: *"Too far, feeling lazy."*
        - AI adapts: *"Try SATCo (closer to Vanderbilt)."*

Why Flipped Interaction Works?

1. Mimics Human Problem-Solving

- AI behaves like a **human assistant**, guiding rather than waiting for commands.

2. Flexibility in Input/Output

- No need for **strictly formatted queries**—AI understands natural language.
- Example:
    - User: *"It's raining, so probably not outside."*
    - AI infers: *"Prioritize indoor seating."*

3. Scalable to Automated Systems

- Same pattern applies when AI interacts with:
    - **Databases** (queries in SQL/API calls).
    - **Other software** (structured commands).
- AI acts as a **"protocol droid"** (like Star Wars' C-3PO), translating goals into system-specific actions.

Applications of Flipped Interaction

1. **Decision Support Systems**
    - Example: Restaurant recommendations, travel planning.

2. **Automated Workflows**
    - AI orchestrates tasks across tools (e.g., CRM, spreadsheets).
3. **Customer Service Chatbots**
    - Proactively guides users through troubleshooting.

Challenges & Considerations

1. **Human-in-the-Loop vs. Full Automation**
    - Humans adapt easily to free-form AI instructions.
    - Machines require **structured inputs** (e.g., API calls).
    - **Solution:** AI must **translate goals** into machine-readable formats.
2. **Error Handling**
    - AI must recover gracefully from unexpected inputs (e.g., *"No veggies, but I have durian"*).
3. **Prompt Engineering**
    - Clear initial prompts are critical (e.g., *"Ask the first question"*).

Key Takeaways

1. **Flipped Interaction = AI drives the conversation.**
2. **Works via iterative Q&A or task delegation.**
3. **Enables adaptability in dynamic environments.**
4. **Foundation for integrating AI with rigid systems (databases, APIs).**

The Agent Loop

From Manual Interaction to Autonomous Agents

Key Problem

How do we transition from:

- **Human-driven conversations** (e.g., manually entering responses) →
- **Autonomous AI agents** that interact with systems without human intervention?

Solution: Implement an **Agent Loop**—a programmatic cycle where the AI:

1. Decides actions autonomously.
2. Executes them via APIs/tools.
3. Adapts based on feedback.

Core Concept: The Agent Loop

1. Human Initiates the Task

- Example: *"Add this travel expense to the spreadsheet."*
- The AI translates **natural language goals → computational actions**.

2. The Loop Structure

The agent operates in a **cycle** until the task is complete:

Step	Description
1. Construct Prompt	Build a prompt specifying the task and available actions (e.g., API calls).
2. Send to LLM	The LLM generates a response (e.g., *"Query the spreadsheet for existing entries."*).

Step	Description
3. Parse Response	Extract the **action** (e.g., get_rows) and **parameters** (e.g., date=2024-07-29).
4. Execute Action	Programmatically run the action (e.g., call a spreadsheet API).
5. Collect Feedback	Capture results/errors (e.g., *"No matching rows found."*).
6. Update Prompt	Feed results back into the next iteration (e.g., *"No rows exist. Add new row?"*).

3. Termination

The loop stops when:

- The task is completed.
- An error requires human intervention.
- A predefined limit (e.g., max iterations) is reached.

Why the Agent Loop Works

1. Adaptability

- Unlike brittle traditional code, the AI **adjusts actions dynamically** (e.g., *"No beans? Use lentils instead."*).
- Handles **unexpected inputs** (e.g., API errors, missing data).

2. Autonomy

- Removes the **human-in-the-loop** for repetitive tasks.
- Translates **goals → actions** without manual coding.

3. Scalability

- Integrates with **APIs, databases, or other tools** (e.g., spreadsheets, CRMs).
- Acts as a **"bridge"** between natural language and rigid systems.

Key Components in Detail

1. Prompt Engineering

- **Critical for success**: The prompt must:
    - Define the **task** clearly.
    - List **available actions/tools** (e.g., get_rows, add_expense).
    - Specify **response format** (e.g., JSON for easy parsing).

2. Parsing Responses

- Challenge: LLMs output **natural language**; systems need **structured data**.
- Solution:
    - Use **delimited formats** (e.g., *"ACTION: get_rows {date: '2024-07-29'}"*).
    - Leverage **function-calling** (e.g., OpenAI's tools parameter).

3. Execution & Feedback

- **Actions**: API calls, file operations, database queries.
- **Feedback**: Success messages, error codes, or data results.
    - Example: *"API_ERROR: Invalid date format."* → AI retries with corrected input.

Example: Travel Expense Agent

Task: *"Add a $50 taxi expense for July 29."*

1. **Iteration 1**:
    - AI: *"Check if expense exists. ACTION: get_rows(date='2024-07-29')"*
    - System: Returns [] (no rows).

2. **Iteration 2**:
    - AI: *"Add new row. ACTION: add_row(amount=50, category='taxi')"*
    - System: Confirms SUCCESS.
3. **Loop Ends**.

Challenges & Solutions

Challenge	Solution
Unstructured LLM outputs	Constrain responses with templates/grammars.
API errors	Retry/adapt using feedback (e.g., *"Invalid date → correct format."*).
Infinite loops	Set max iterations or timeout limits.

Key Takeaways

1. **Agent Loop = Autonomous task execution cycle.**
2. **Steps**: Prompt → LLM → Parse → Execute → Feedback → Repeat.
3. **Critical for**: Adaptability, scalability, and human-free automation.
4. **Next Step**: Implement the loop in Python with API integrations.

These sections are written in a Colab Notebook:
*** Sending Prompts Programmatically & Managing Memory**
*** Giving Agents Memory**
*** Building a Quasi-Agent**
*** Building a Simple AI Agent**

[Link of the Colab Notebook (Github)]

Adding Structure to AI Agent Outputs

The Core Challenge

Generative AI models are:

- **Non-deterministic**: Outputs vary across runs.
- **Verbose**: Include explanations, alternatives, and markdown.
- **Unstructured**: No consistent format for programmatic parsing.

Problem:s
To build autonomous agents, we need **structured outputs** that:

1. Clearly specify **actions** (e.g., API calls, function executions).
2. Can be **parsed reliably** by traditional software.

Two Approaches to Structured Outputs

1. **Prompt Engineering + Parsing** (Works with any LLM)
2. **Function Calling** (Requires LLM support, e.g., OpenAI's tools parameter)

This lecture focuses on **Approach 1**: Designing prompts to enforce consistent action formats.

Method: Template-Based Prompt Engineering

Key Idea

Define a **strict output template** in the prompt, including:

- **Placeholders** for actions (e.g., ACTION: <action>).
- **Separation** of reasoning (chatty) from actions (structured).

Example 1: Cooking Agent

Prompt:

```
Actions: [pickup, use, discard]

Objects: [pan, butter, green beans, salt, garlic, spatula]

Output format: ACTION:object

Example: pickup:pan

Output ONE action at a time.
```

User Request: *"Cook savory green beans."*
AI Output: *pickup:pan*

Feedback Loop:

- User: *"Handle breaks off pan."*
- AI Adapts: *discard:pan*

Observations:

- The LLM adheres to the ACTION:object template.
- **Variations exist** (e.g., capitalization: PICKUP vs. pickup), requiring flexible parsing.

Example 2: Web Scraping Agent

Prompt:

```
Actions:
- fetch_web_page_text(url)
- base64_encode(text)

Output format:
REASONING: <your reasoning>
ACTION: var = action(params)
```

User Request: *"Encode Vanderbilt's homepage in base64."*
AI Output:
REASONING: First fetch the page, then encode it.
ACTION: page_text = fetch_web_page_text("https://vanderbilt.edu")

Feedback Loop:

- User: *"page_text = [HTML content]."*

AI: *ACTION: encoded_text = base64_encode(page_text)*

Advantages:

- **Programmable outputs**: Resemble function calls (e.g., fetch_web_page_text(url)).
- **Clear separation**: Reasoning (verbose) vs. actions (structured).

Prompt Engineering vs Function Calling - Which to Choose?

When building AI agents, you've got two main ways to get structured outputs from LLMs:

1. **Prompt Engineering**
    - Works with any LLM
    - You design custom output formats (like "ACTION: do_thing")
    - More flexible for unique needs
    - Best when:
        - Your LLM doesn't support fancy features
        - You need special command formats
        - You're doing something unusual
2. **Function Calling**
    - Only works with certain LLMs (like OpenAI)
    - Gives you clean JSON responses automatically
    - Less setup work
    - Best for:
        - Standard API integrations
        - When you want ready-to-use structured data

Key Things to Remember:

- Structure is everything - your agent needs clear actions to follow
- Good prompts create consistency (templates are your friend)
- The feedback loop is what makes agents smart and adaptable
- There's always a tradeoff: more control vs easier setup

- **This section is written in a Colab Notebook:**
  *** Building a Simple AI Agent**

Link of the Colab Notebook (Github)