AI Agents: Intro Notes

Introduction to AI Agents

An AI agent is an autonomous system powered by an Artificial Intelligence (AI) model, typically a Large Language Model (LLM), designed to interact with its environment to achieve user-defined objectives. By combining reasoning, planning, and action execution, often through external tools, AI agents perform tasks effectively and dynamically.

Components of an AI Agent

An AI agent consists of two primary components:
- The Brain (AI Model): The core reasoning engine, usually an LLM, responsible for understanding instructions, analyzing data, planning, and deciding on actions.
- The Body (Capabilities and Tools): The set of actions an agent can perform, defined by its equipped tools or interfaces. For example, just as humans cannot fly due to a lack of wings, an AI agent's actions are limited by its available tools.

Core Capabilities

AI agents are designed to:
- Understand Natural Language: Interpret and respond to human instructions in a conversational manner.
- Reason and Plan: Analyze information, strategize, and break down complex tasks into manageable steps.
- Interact with the Environment: Gather data, execute actions (e.g., via APIs), and observe outcomes to refine future decisions.

Large Language Models (LLMs): The Core of AI Agents

What is an LLM?

A Large Language Model (LLM) is an AI model trained on extensive text datasets to understand and generate human-like language. With millions to billions of parameters, LLMs capture linguistic patterns, structures, and nuances, making them the backbone of AI agents.

Transformer Architectures

LLMs are built on the Transformer architecture, which has three main variants:

1. Encoders
   - Function: Transform input text into dense representations (embeddings).
   - Example: BERT (Google).
   - Use Cases: Text classification, semantic search, named entity recognition.
   - Typical Size: Millions of parameters.
2. Decoders
   - Function: Generate text by predicting the next token in a sequence.
   - Example: Llama (Meta).
   - Use Cases: Text generation, chatbots, code completion.
   - Typical Size: Billions of parameters.
3. Sequence-to-Sequence (Encoder-Decoder)
   - Function: Process input with an encoder and generate output with a decoder.
   - Examples: T5, BART.
   - Use Cases: Machine translation, summarization, paraphrasing.
   - Typical Size: Millions to billions of parameters.

How LLMs Work

LLMs are autoregressive, predicting the next token in a sequence based on prior tokens until reaching an End of Sequence (EOS) token. The Attention mechanism prioritizes relevant tokens (e.g., "France" and "capital" in "The capital of France is...") to generate coherent outputs.

Messages and Chat Templates

Messages in Conversational AI

In conversational AI systems (e.g., ChatGPT), user interactions are structured as messages:
- System Messages: Define the model’s behavior (e.g., "Act as a technical tutor"). These persist across interactions.
- User Messages: Inputs from the human, such as questions or commands.
- Assistant Messages: Responses generated by the LLM.

These messages are concatenated into a single prompt for the LLM to process.

Chat Templates

Chat templates format messages to maintain conversation history, ensuring coherent multi-turn interactions. For example:
System: You are a helpful assistant.
User: What is the capital of France?
Assistant: The capital of France is Paris.
User: What's the weather there?
Assistant: I'll check the weather for Paris using a tool.

Base Models vs. Instruct Models

- Base Models: Trained on raw text for next-token prediction (e.g., SmolLM2-135M). They lack conversational finesse.
- Instruct Models: Fine-tuned to follow instructions and engage in dialogue (e.g., SmolLM2-135M-Instruct).
To make a base model act like an instruct model, prompts must be consistently formatted using chat templates.

AI Tools: Enhancing Agent Functionality

What Are AI Tools?

AI tools are functions that extend an LLM's capabilities, enabling tasks like data retrieval, calculations, or API calls. Since LLMs only process text, tools act as interfaces between the model and the environment.

How Tools Work

1. Tool Awareness: The LLM is informed about available tools via descriptions (e.g., "weather_tool: Fetches weather data for a location").
2. Tool Invocation: The LLM generates a text-based command, such as `call weather_tool('Paris')`, when prompted.

3. Execution: The agent’s framework executes the tool and returns the result (e.g., "Sunny, 20°C").
4. Integration: The result is appended to the LLM’s context for further reasoning.

Example: Generic Tool Implementation

Below is a Python class for a reusable tool, inspired by real-world libraries:

```
from typing import Callable

class Tool:
    """
    A class representing a reusable tool for AI agents.

    Attributes:
        name (str): Name of the tool.
        description (str): Description of the tool's purpose.
        func (Callable): The function the tool executes.
        arguments (list): List of argument names and types.
        outputs (str): The return type(s) of the function.
    """
    def __init__(self, name: str, description: str, func: Callable, arguments: list, outputs: str):
        self.name = name
        self.description = description
        self.func = func
        self.arguments = arguments
        self.outputs = outputs

    def to_string(self) -> str:
        """
        Returns a string representation of the tool.
        """
        args_str = ", ".join([f"{arg_name}: {arg_type}" for arg_name, arg_type in self.arguments])
        return (
            f"Tool Name: {self.name}, "
            f"Description: {self.description}, "
            f"Arguments: {args_str}, "
            f"Outputs: {self.outputs}"
        )

    def __call__(self, *args, **kwargs):
        """
        Invokes the underlying function with the provided arguments.
        """
        return self.func(*args, **kwargs)
```

This class:
- Stores tool metadata (name, description, arguments, outputs).
- Provides `to_string()` for LLM-readable descriptions.
- Enables function execution via `__call__()`.

Importance of Tools

- Access real-time data (e.g., weather APIs).
- Perform specialized tasks (e.g., database queries).
- Overcome limitations of static LLM training.

The Thought-Action-Observation Cycle

AI agents operate in a cyclical process of Thought, Action, and Observation, continuing until the objective is met.

Components of the Cycle

1. Thought: The LLM reasons about the task, analyzes context, and plans the next step using approaches like ReAct (Reasoning + Acting).
2. Action: The agent executes a step, such as calling a tool (e.g., `{ "tool": "weather_tool", "args": ["Paris"] }`).
3. Observation: The agent receives feedback (e.g., API responses, errors) and integrates it into its context.

How the Cycle Operates

- Thought: The LLM evaluates the task and prior observations to decide on an action.
- Action: The agent invokes a tool or performs an operation.
- Observation: The result is parsed and appended to the context, informing the next cycle.

Example: Weather Query

Consider an agent, Alfred, tasked with answering, "What's the weather in Paris?"
1. Thought: Alfred identifies the need for real-time data and selects `weather_tool`.
2. Action: Alfred generates `call weather_tool('Paris')`.
3. Observation: The tool returns "Sunny, 20°C," which Alfred appends to its context.
4. Next Cycle: If the result is incomplete, Alfred retries or adjusts its approach.

Dynamic Adaptation

- Incorporates new information into reasoning.
- Refines strategies based on feedback.
- Ensures accurate, context-aware responses.

Conclusion

AI agents, powered by LLMs and enhanced by tools, are dynamic systems that bridge human instructions with environmental interactions. Through the Thought-Action-Observation cycle, agents solve complex problems, access real-time data, and adapt to changing conditions. Tools and chat templates further enhance their capabilities, making them versatile for applications like automation, customer service, and information retrieval.