Whitepaper

The Architecture Behind State-of-the-Art AI Agents

1. Overview

Modern AI agents combine large language models with planning, memory and tool integration to operate autonomously on complex goals. They are not single black-box models. They are modular, interactive systems that loop reasoning, action and memory together. ([NVIDIA Developer][1])

2. Core Architecture (Single-Agent Pattern)

Here’s the high-level architecture of a typical single AI agent:

```
                   +---------------------+
                   |  User / Environment |
                   +---------+-----------+
                             |
                             v
                     +----------------+
                     | Input Layer    |
                     | (Parsing/Pre   |
                     |  Processing)   |
                     +------+---------+
                            |
                            v
             +--------------------------------+
             |           Agent Core           |
             | (LLM + Planner + Controller)   |
             +-----+-----------+--------------+
                   |           |
      +------------+           +-----------------+
      |                                          |
      v                                          v
+--------------+                       +-------------------+
|   Memory     |  <--context queries   |    Tools / APIs   |
|(Short & Long)|                       | (DB, Search, Code)|
+------+-------+                       +-----------+-------+
       ^                                           |
       |                                           v
       +> Output or Action
```

How it works:

1. **Input Layer:** Parses user requests and environment data such as text, webhook events, files, or sensor input. ([Designveloper][2])
2. **Agent Core:** The LLM serves as a reasoning engine that the planner uses to break goals into steps. It calls the memory and tools as needed. ([NVIDIA Developer][1])

3. **Memory:** Provides context back into the loop so the agent can “remember” past actions or facts. ([Akka][3])
4. **Tools / APIs:** These are external functions that perform actions like querying a database, running code, calling search, scheduling meetings, or invoking other services. ([Leanware][4])
5. **Output / Action:** Final results are either sent back to the user or used to enact changes in external systems.

This design combines reasoning, memory and tool execution into a single loop, with the LLM orchestrating the whole process.

3. Planning + Reasoning Loop

A key part of any agent is the planning and acting loop. A common pattern is called **ReAct (Reason + Act)**. In simple form:

```
 User Goal
    |
    v
+------------------------------+
| LLM Planning & Reasoning     |
|   "What should I do first?"  |
+---+--------------------------+
    |
    v
+------------------------------+
| Action Step (Call Tool)      |
+------------------------------+
    |
    v
+------------------------------+
| Observe Output               |
| Send back into next planning |
+------------------------------+
    |
    v
Repeat or loop
```

In this loop:

- The agent uses the model to decide the next action.
- It calls a tool.
- It reads the result.
- It re-plans based on new context. This continues until the goal is reached. ([Redis][5])

4. Memory Integration

Memory gives agents continuity over time. It can be as simple as keeping recent conversation context, or as complex as long-term knowledge bases. Two types are common:

```
    +-------------------+
    |   Memory Store    |
    |                   |
    | Short Term        |
    | (Session Context)|
    |                   |
    | Long Term         |
    | (Knowledge Graph,|
    | Persistent Store)|
    +--------+---------+
             |
          Retrieve
             |
             v
       +-------------+
       |   LLM Core  |
       +-------------+
```

Algorithms like semantic similarity or vector indexing are used to pull relevant past interactions into the reasoning context.

5. Tool Integration Stack

A powerful agent doesn’t just generate text. It interacts with the world.

```
+---------------------+
| Tool Orchestrator   |
|---------------------|
| API Adapters        |
| Database Connectors |
| Function Executors  |
| Custom Plugins      |
+----+----+----+------+
     |    |    |
     v    v    v
  Search  Code  Data
  API     Runner Queries
```

Agents decide what tool to call and in what order. This makes them capable of scheduling a meeting, analysing code, querying a database, or scraping a website. ([Leanware][4])

6. Multi-Agent Architectures

For complex tasks, agents can be organised into networks:

```
     +----------------------+
     |   Orchestrator /    |
     |   Supervisor Agent  |
      +---------+---------+
                |
     +----------+-----------+
     |          |           |
     v          v           v
+--------+  +-----------+ +---------+
| Planner|  | Retriever | | Executor|
+--------+  +-----------+ +---------+
```

In multi-agent setups:

- A **supervisor agent** delegates subtasks.
- Specialist agents handle retrieval, reasoning, execution, or quality checks.
- Communication between agents follows protocols and message formats. ([Speakeasy][6])

This is useful for workflows that need parallelism or domain-specific expertise.

7. Typical Runtime Data Flow

Here's a simplified sequence when an agent runs:

```
User Goal
     |
     v
Parse + Context Add
     |
     v
LLM Plans Steps --> Add to Memory
     |
     v
Call Tool 1 --> Return Output
     |
     v
Update Context + Memory
     |
     v
LLM Plans Next Step
     |
     v
...
     |
     v
Final Answer
```

At each loop, the agent updates memory and reevaluates the next action. This loop gives the agent flexibility and adapts to new work as it arrives.

8. Practical Frameworks & Protocols

To implement this in real systems, frameworks such as LangChain, AutoGen, or event protocols like the **Model Context Protocol (MCP)** are used to standardise how agents talk to tools. MCP, for example, provides a JSON-RPC interface for tool execution and data exchange with LLMs.

These standards help enforce consistent interfaces so agents can scale, integrate with different services, and avoid ad-hoc point-to-point code.

9. Limitations & Where Architects Must Focus

Even with solid diagrams and modules, AI agents are not magic:

- They still rely on the underlying model's reasoning quality.
- Memory systems need careful curation to avoid contradictions over time.
- Tool invocation logic must be guarded to prevent unsafe actions.
- Multi-agent coordination adds complexity that requires robust supervision logic.

Good architecture is not just splitting components up. It's also about clear contracts between them and disciplined state management.

10. Conclusion

Modern AI agents are engineered systems built around LLM reasoning, memory, planning and external tools. The diagrams above show how those pieces fit together and illustrate why these agents can do much more than just generate text.

They are planning, acting, remembering and adjusting systems that interact with APIs and environments. Understanding the architectural patterns helps you design agents that are reliable, scalable and fit for real-world use.

References

1. https://developer.nvidia.com/blog/introduction-to-llm-agents/?utm_source=chatgpt.com "Introduction to LLM Agents | NVIDIA Technical Blog"
2. https://www.designveloper.com/blog/ai-agent-architecture-diagram/?utm_source=chatgpt.com "Guide to AI Agent Architecture with Diagrams"
3. https://akka.io/blog/agentic-ai-architecture?utm_source=chatgpt.com "Agentic AI architecture 101: An enterprise guide"
4. https://www.leanware.co/insights/llm-agent-architecture-guide?utm_source=chatgpt.com "LLM Agent Architecture: A Complete Guide - Leanware"
5. https://redis.io/blog/ai-agent-architecture-patterns/?utm_source=chatgpt.com "AI Agent Architecture Patterns: Single & Multi-Agent Systems"
6. https://www.speakeasy.com/mcp/using-mcp/ai-agents/architecture-patterns?utm_source=chatgpt.com "A practical guide to the architectures of agentic applications"
7. https://en.wikipedia.org/wiki/Model_Context_Protocol?utm_source=chatgpt.com "Model Context Protocol"
8. https://www.decodingai.com/p/getting-agent-architecture-right?utm_source=chatgpt.com "Getting Agent Architecture Right"