L3-M2: Anatomy of an AI Agent - Architecture and Components

Module ID: L3-M2 **Author:** Manus AI **Date:** October 30, 2025

I. Introduction to Agentic AI Systems

1.1. Defining the AI Agent

The concept of an **Artificial Intelligence Agent** (AI Agent) represents a paradigm shift from static, reactive models to dynamic, autonomous computational entities. Formally, an AI Agent is any entity that perceives its environment through sensors and acts upon that environment through actuators, exhibiting a degree of **autonomy** and **goal-directed behavior** [1]. Unlike a traditional Large Language Model (LLM) which is a function that maps an input prompt to an output completion, an AI Agent is a system that uses an LLM as its central **Inference Engine** to iteratively plan, reason, and execute actions to achieve a complex, multi-step objective.

The defining characteristic of an agent is its ability to operate in a continuous loop, making decisions based on its current state, memory, and environmental feedback. This iterative process allows agents to tackle tasks that are too complex, too long, or too uncertain for a single, monolithic LLM call. The evolution from simple symbolic AI agents to modern LLM-powered agents is marked by the integration of sophisticated natural language understanding and generation capabilities with external tools and persistent memory systems.

1.2. The Need for Agent Architectures

The inherent limitations of monolithic LLMs necessitate a structured agent architecture. These limitations primarily revolve around the **context window constraint** and the **lack of external interaction**. A single LLM call cannot reliably manage a long, multi-step process, nor can it directly interact with the real world (e.g., browsing the web, running code, accessing databases).

An agent architecture provides the necessary scaffolding to overcome these challenges. It modularizes the complex task of intelligence into distinct, manageable components: perception, memory, reasoning, and action. This modularity allows the system to:

1. **Extend Capabilities:** By integrating external tools (actuators), the agent can perform actions outside the LLM's intrinsic capabilities.
2. **Maintain State:** Persistent memory systems allow the agent to recall past interactions and knowledge, overcoming the LLM's stateless nature.
3. **Ensure Reliability:** The iterative loop structure enables self-correction and dynamic replanning based on real-time feedback.

The architecture transforms the LLM from a mere text generator into the **central processing unit** of a more comprehensive, problem-solving system.

II. The Core Architectural Components

The modern AI Agent architecture is typically composed of four primary, interconnected components that facilitate the continuous cycle of intelligent behavior.

2.1. The Perception Component

The **Perception Component** is the agent's interface to its environment. It is responsible for gathering and interpreting all relevant information, translating raw environmental data into a structured format that the LLM can process. This structured input is often referred to as the **Observation**.

Perception is not limited to a single modality. Inputs can include:

- **Textual Data:** User prompts, file contents, search results, API documentation.
- **Structured Data:** JSON or XML responses from external services.
- **Tool Outputs:** The result of a code execution, a database query, or a web scrape.
- **Multimodal Data:** Transcribed speech, image descriptions, or sensor readings (often pre-processed by specialized models before reaching the central LLM).

The quality and completeness of the Observation are critical, as they directly influence the subsequent reasoning and action steps. A robust perception component must also

include mechanisms for filtering and prioritizing information to prevent context window overflow.

2.2. The Memory System

Memory is essential for an agent to exhibit coherent, long-term behavior and to learn from experience. It is typically segmented into two primary types:

Short-Term Memory (Context Window)

This is the agent's immediate operational memory, often implemented as the LLM's **context window** itself. It acts as a **scratchpad** where the agent stores the current conversation history, the immediate goal, the reasoning trace (Thought), the last action taken (Action), and the result of that action (Observation). This memory is transient and primarily serves the current, ongoing task. The size of this memory is a major constraint in complex tasks, as the agent must constantly manage which information to keep and which to discard.

Long-Term Memory (Knowledge Base)

This memory is persistent and stores information beyond the current context window, allowing the agent to recall information from past tasks or a vast external knowledge base. This is commonly implemented using **Retrieval-Augmented Generation (RAG)** techniques, where external documents, user preferences, or past experiences are stored in a **vector database** [2]. When the agent needs information, a query is sent to the vector database, and the most relevant chunks of text are retrieved and inserted into the LLM's context window.

Memory Type	Implementation	Purpose	Lifespan	Key Challenge
Short-Term	LLM Context Window (Scratchpad)	Immediate task execution, tracking current state.	Transient (per-task loop)	Context Window Overflow
Long-Term	Vector Databases, Knowledge Graphs	Storing and retrieving vast, persistent knowledge.	Persistent (across tasks)	Retrieval Relevance

2.3. The Reasoning/Planning Component (The Brain)

The **Reasoning Component**, powered by the LLM, is the "brain" of the agent. Its primary function is to process the current Observation, access memory, determine the current state, and formulate the next step required to move closer to the overall goal.

This component employs sophisticated reasoning techniques to enhance the LLM's raw capability:

- **Chain-of-Thought (CoT):** The LLM is prompted to generate a step-by-step reasoning trace before providing a final answer or action. This improves logical consistency and task decomposition.
- **Planning:** For complex goals, the agent first generates a high-level plan (e.g., a sequence of sub-goals). During execution, the reasoning component constantly checks the current state against the plan and adjusts the next action accordingly.
- **Self-Correction:** By analyzing the Observation resulting from a previous Action, the reasoning component can identify errors or unexpected outcomes and generate a new plan or action to correct the trajectory.

2.4. The Action/Tool-Use Component (The Hands)

The **Action Component** allows the agent to execute the decisions made by the reasoning component, enabling interaction with the external environment. This is often achieved through **Tool-Use** or **Function Calling**, where the LLM's output is not a direct answer but a structured call to an external function or API.

The process involves:

1. **Tool Selection:** The reasoning component determines which of the available tools (e.g., web search, code interpreter, calculator, database query) is most appropriate for the current step.
2. **Parameter Generation:** The LLM generates the necessary input arguments for the selected tool.
3. **Tool Invocation:** The agent's executive layer executes the function call.
4. **Observation Return:** The tool's output is captured and fed back into the Perception component as the next Observation, closing the loop.

This component is crucial for grounding the agent in reality, allowing it to access up-to-date information and perform state-changing operations.

III. The Foundational Agent Cycle: Perception-Reasoning-Action (P-R-A)

The **Perception-Reasoning-Action (P-R-A) cycle** is the fundamental control loop governing the behavior of nearly all autonomous AI Agents [3]. It is a continuous, iterative process that allows the agent to adapt to a dynamic environment and progress toward its goal.

3.1. Overview of the P-R-A Cycle

The cycle can be formally described as a sequence of three phases, repeated until the goal is achieved or deemed impossible:

1. Perceive (Observation)

The agent senses the environment and receives feedback from its last action. This input, the **Observation**, updates the agent's internal model of the world.

2. Reason (Thought)

The agent processes the Observation, consults its memory, and uses the LLM to generate a **Thought**. This thought is an internal monologue that includes: * Analyzing the Observation. * Updating the current goal state. * Determining the next logical step. * Selecting the appropriate tool or action.

3. Act (Action)

Based on the Thought, the agent executes a physical or digital **Action**. This might be generating a final answer, calling an external tool, or modifying a file. The action changes the state of the environment or the agent itself.

After the Action is executed, the cycle immediately returns to the **Perceive** phase to observe the result of the action, creating a closed-loop system. This continuous feedback mechanism is what grants the agent its robustness and ability to self-correct.

3.2. Practical Application of P-R-A

Consider an agent tasked with **"Finding the current stock price of Tesla and summarizing the last three news headlines about the company."**

Step	Phase	Agent Activity	Outcome/Observation
1	**Perceive**	Receives initial user prompt and goal.	Observation: "Find stock price and summarize news."
2	**Reason**	*Thought*: I need two pieces of information. First, I will use the `stock_price_tool` for the current price.	Action: `stock_price_tool(symbol="TSLA")`
3	**Act**	Executes the `stock_price_tool`.	Observation: "Current Price: $250.15"
4	**Reason**	*Thought*: I have the price. Now I need the news. I will use the `news_search_tool` for the last three headlines.	Action: `news_search_tool(query="Tesla news", count=3)`
5	**Act**	Executes the `news_search_tool`.	Observation: "Headline 1: New Gigafactory announced. Headline 2: Q3 Earnings Beat. Headline 3: Autopilot Safety Review."
6	**Reason**	*Thought*: I have all required information. I will now synthesize and present the final answer to the user.	Action: `final_answer("The current stock price is $250.15. Recent news...")`

This example illustrates how the agent iteratively uses the P-R-A cycle to break down a complex goal, use external resources, and synthesize a final response.

IV. Deep Dive: The ReAct Framework

4.1. Introduction to ReAct (Reasoning and Acting)

ReAct is an architectural framework proposed in 2022 by Yao et al. that explicitly synergizes the concepts of **Reasoning** (via Chain-of-Thought) and **Acting** (via tool-use) [4]. The name ReAct is a portmanteau of **Re**asoning and **Act**ing. Its core innovation lies in **interleaving** the generation of an internal reasoning trace (Thought) with the execution of task-specific actions (Action) and the subsequent environmental feedback (Observation).

Prior to ReAct, systems typically relied on either purely reasoning (CoT) or purely action-based approaches. CoT could solve complex logical problems but could not interact with the world, while action-based models often lacked the transparency and planning capability needed for multi-step tasks. ReAct combines the strengths of both, allowing the LLM to dynamically plan and adapt its actions based on the outcomes it observes.

4.2. The ReAct Loop Explained (Step-by-Step)

The ReAct loop is a formalized version of the P-R-A cycle, making the internal **Thought** explicit and central to the process. The agent's prompt template is structured to guide the LLM to generate a sequence of three distinct tags: `Thought`, `Action`, and `Observation`.

The loop proceeds as follows:

1. **Input:** The agent receives the user's prompt and the current history of the ReAct trace (if any).
2. **Thought (Reasoning):** The LLM generates a **Thought**. This is an internal monologue where the agent analyzes the context, reflects on the last Observation, and determines the strategic next step. This step is crucial for planning, error correction, and maintaining transparency.
3. **Action (Acting):** Based on the Thought, the LLM generates an **Action** in a predefined format (e.g., `Action: [tool_name[tool_input]]`). This action is then passed to the execution environment.

4. **Observation (Perception):** The execution environment runs the tool specified in the Action and returns the result as an **Observation**. This observation is then appended to the prompt history.
5. **Iteration:** The entire sequence (Thought, Action, Observation) is repeated. The LLM now processes the *entire* history, including the new Observation, to generate the *next* Thought, allowing the agent to continuously refine its plan until the final answer is ready.

Example ReAct Trace

The following is a simplified trace for an agent using a `Search` tool to answer the query: "Who was the first person to walk on the moon, and what year did they graduate from Purdue University?"

Step	Type	Content
1	**Thought**	I need to identify the first person to walk on the moon. I will use the Search tool for this initial fact.
2	**Action**	`Search[first person to walk on the moon]`
3	**Observation**	Neil Armstrong.
4	**Thought**	I have the name: Neil Armstrong. Now I need to find the year he graduated from Purdue University.
5	**Action**	`Search[Neil Armstrong Purdue University graduation year]`
6	**Observation**	1955 (Aeronautical Engineering).
7	**Thought**	I have both pieces of information. I will now synthesize the final answer.
8	**Action**	`Final Answer[Neil Armstrong was the first person to walk on the moon, and he graduated from Purdue University in 1955 with a degree in Aeronautical Engineering.]`

4.3. Advantages and Limitations of ReAct

The ReAct framework offers significant improvements over simpler agent designs, but it is not without its trade-offs.

Feature	ReAct Framework	Traditional CoT (No Tools)	Simple Tool-Use (No CoT)
Reasoning	Explicit, Interleaved (Thought)	Explicit, Sequential	Implicit/Minimal
Tool Use	Fully Integrated	None	Integrated, but less strategic
Transparency	High (Traceable Thoughts)	High (Traceable Steps)	Low (Black Box Action)
Self-Correction	High (Can correct based on Observation)	Low (Cannot correct external failures)	Moderate (Can retry actions)
Latency	High (Multiple LLM calls)	Moderate (One long LLM call)	Low (Fewer LLM calls)
Context Pressure	High (Trace grows with each step)	Moderate	Low

The primary advantage of ReAct is its **increased reliability** and **transparency**. By forcing the LLM to externalize its reasoning, the agent can more effectively debug its own process and adapt to unexpected tool outputs. The main limitation is the **increased latency** and **context window pressure** due to the necessity of passing the entire, growing trace back to the LLM at every step.

V. Advanced Agent Concepts and Future Directions

5.1. Multi-Agent Systems (MAS)

As tasks become more complex, a single agent may be insufficient. **Multi-Agent Systems (MAS)** involve multiple specialized AI agents collaborating to achieve a shared, overarching goal [5]. In an MAS, each agent is typically assigned a distinct role (e.g., Planner, Coder, Reviewer, Researcher), and they communicate with each other using structured protocols.

This architecture mimics human organizational structures and offers several benefits:

- **Specialization:** Agents can be optimized for specific tasks, such as one agent dedicated to web browsing and another to code generation.
- **Parallelization:** Sub-tasks can be executed simultaneously by different agents, accelerating the overall process.
- **Robustness:** Failure in one agent does not necessarily halt the entire system, as other agents can often compensate or take over.

A common real-world example is an AI-powered software development team, where a "Product Manager Agent" defines the task, a "Developer Agent" writes the code, and a "Testing Agent" verifies the output, all communicating via a shared memory or message queue.

5.2. Ethical and Safety Considerations

The autonomy inherent in advanced agent architectures, particularly those utilizing continuous P-R-A or ReAct loops, introduces significant **ethical and safety challenges**.

The core concern is **Agent Alignment**, ensuring that the agent's actions and sub-goals remain aligned with the user's intent and societal values. Since agents can autonomously select tools and generate new plans, the potential for unintended or harmful actions increases.

Architectural safeguards are necessary to mitigate these risks:

- **Action Constraints:** Implementing strict, predefined limits on the tools an agent can call (e.g., blocking access to sensitive APIs or file systems).
- **Observation Filtering:** Employing a "safety layer" model to review tool outputs and environmental feedback for malicious or harmful content before it reaches the main LLM.
- **Human-in-the-Loop (HITL):** Designing the loop to require human approval for high-risk actions, such as deploying code or making financial transactions.

The development of robust, safe, and transparent agent architectures is an active and critical area of research, focused on balancing the agent's autonomy with necessary control mechanisms.

VI. Conclusion

6.1. Key Takeaways

The transition from static, single-turn LLMs to dynamic, autonomous AI Agents is fundamentally driven by sophisticated architectural design. The **Anatomy of an AI Agent** is defined by the synergistic interaction of its four core components: **Perception**, **Memory**, **Reasoning**, and **Action**.

The continuous **Perception-Reasoning-Action (P-R-A) cycle** provides the operational framework for autonomy, allowing the agent to continuously adapt and progress toward complex goals. The **ReAct framework** formalizes this cycle by explicitly interleaving **Thought** (reasoning) and **Action** (tool-use), leading to more reliable, transparent, and self-correcting behavior.

The future of AI systems lies in these dynamic, architectural approaches, moving the field beyond simple models to deployable, robust, and goal-directed intelligent systems capable of solving real-world problems.

6.2. Further Reading and References

[1] Russell, S. J., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson Education. [2] Lewis, P., et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems (NeurIPS)*. [3] IBM. (n.d.). *Components of AI Agents*. Retrieved from https://www.ibm.com/think/topics/components-of-ai-agents [4] Yao, S., et al. (2022). ReAct: Synergizing Reasoning and Acting in Language Models. *International Conference on Learning Representations (ICLR)*. Retrieved from https://arxiv.org/abs/2210.03629 [5] Gu, J., et al. (2024). A Survey of Large Language Model-based Multi-Agent Systems. *arXiv preprint arXiv:2402.09888*. Retrieved from https://arxiv.org/abs/2402.09888 [6] Markovate. (2024). *Agentic AI Architecture: A Deep Dive*. Retrieved from https://markovate.com/blog/agentic-ai-architecture/ [7] Google Research. (2022). *ReAct: Synergizing Reasoning and Acting in Language Models*. Retrieved from https://research.google/blog/react-synergizing-reasoning-and-acting-in-language-models/