2026 Volume: 02 Issue: 04 www.ijrpa.com ISSN 2456-9995 Research Article

International Journal Research Publication Analysis

Page: 01-04

AUTONOMOUS AI AGENT USING LLM

[*1]Dr. Ramya BN, [2]Shwetha Patil, [3]Sangeetha M.

[1]Associate Professor, Department of Computer Science and Engineering, Jyothy Institute of Technology, Bengaluru, India.

[2,3]Department of Computer Science and Engineering, Jyothy Institute of Technology, Bengaluru, India.

Article Received: 17 March 2026
Article Revised: 07 April 2026
Published on: 27 April 2026

***Corresponding Author: Dr. Ramya BN**
Associate Professor, Department of Computer Science and Engineering, Jyothy Institute of Technology, Bengaluru, India.

DOI: https://doi-doi.org/101555/ijrpa.8755

ABSTRACT:

Autonomous agents powered by tool integration represent an emerging paradigm in artificial intelligence systems. This paper presents the design and implementation of a deterministic autonomous agent capable of executing predefined Python-based commands and interacting with system tools such as time retrieval and controlled computation. Two agent architectures are explored: a fixed-action deterministic agent and an interactive user-driven agent with extended command capabilities. The system ensures safety by restricting arbitrary code execution and replacing it with controlled command mappings. Experimental demonstrations show that the agent can perform tasks such as random number generation, structured responses, and dynamic command execution while maintaining predictable behavior. This study highlights the importance of controlled tool invocation, modular design, and safety constraints in building reliable autonomous systems modular design, and safety constraints in system.

I. INTRODUCTION:

Autonomous agents are increasingly being integrated into modern artificial intelligence systems to perform tasks independently using reasoning and tool interaction. These agents simulate decision-making by combining internal logic with external tool execution, enabling them to solve problems dynamically. However, traditional AI agents often suffer from unpredictability when executing arbitrary code or interacting with uncontrolled environments. This unpredictability introduces challenges related to safety, reproducibility, and reliability.

To address these concerns, this paper proposes a controlled autonomous agent framework that

restricts execution strictly to predefined commands. By doing so, the system ensures predictable and secure behavior while still maintaining functional flexibility. Two variations of the agent are implemented in this study. The first is a deterministic agent that generates fixed actions regardless of context, while the second is an interactive agent that accepts user inputs and executes corresponding mapped commands. The primary objective of this work is to demonstrate how structured tool usage can significantly improve both safety and usability in autonomous systems.

II.METHODOLOGY

The proposed system is designed using a modular architecture consisting of three core components: the tool layer, the parser module, and the agent core. The tool layer is responsible for executing predefined functions such as random number generation, date and time retrieval, and returning structured responses. The parser module extracts the action and input from the agent's output and ensures that only valid and controlled commands are executed. The agent core is responsible for generating decisions, executing actions via tools, and storing results in memory for traceability.

The deterministic agent implementation follows a fixed reasoning pattern, where it consistently produces the same action and input. Specifically, it always selects a Python-based action with the input "random_number," which generates outputs using a controlled random function. This approach ensures reproducibility and demonstrates the baseline behavior of an autonomous agent operating under strict constraints.

In contrast, the interactive agent enhances system flexibility by allowing user input to determine the agent's behavior. It supports a variety of commands, including random number generation, greetings, date retrieval, coin flips, fruit selection, jokes, square number calculations, and structured JSON outputs. For example, a coin flip command returns either "Heads" or "Tails" using a predefined mapping. This design allows the agent to dynamically respond to user requests while maintaining strict safety boundaries.

A critical aspect of the system is its safety mechanism. Unlike traditional systems that may rely on unrestricted code execution using functions such as eval(), this framework restricts execution strictly to predefined commands. Any unknown or invalid input is handled safely by returning controlled responses instead of executing arbitrary code. This significantly reduces security risks and ensures system robustness.

III. SYSTEM ARCHITECTURE

The system follows a structured data flow beginning with user input or a predefined agent thought. This input is processed by the parser, which extracts the relevant command and validates it. The validated command is then passed to the tool execution layer, where the corresponding function is executed. The output generated by the tool is returned to the agent and stored in memory for future reference. This pipeline ensures a clear and traceable flow of information throughout the system.

The architecture consists of several key components. The input layer accepts either user-provided commands or system-generated instructions. The processing layer parses and validates these inputs to ensure compliance with predefined rules. The execution layer performs the actual computation or response generation using mapped tool functions. Finally, the output layer presents the results to the user, including enhanced readability through features such as colored terminal output.

An integral part of the system is the memory component, which stores both the agent's thoughts and the corresponding results at each step. This enables traceability, debugging, and analysis of agent behavior over time, contributing to improved system transparency.

IV. RESULTS AND DISCUSSION

The functional evaluation of the system highlights clear differences between the deterministic and interactive agents. The deterministic agent consistently produces reproducible outputs due to its fixed execution pattern, while the interactive agent offers greater flexibility by responding dynamically to user inputs. Both agents maintain a high level of safety due to the controlled execution environment.

Observations from the system demonstrate that the deterministic agent is highly reliable for predictable tasks, whereas the interactive agent significantly improves usability and user engagement. The use of command mapping enhances security by preventing unauthorized operations, and the inclusion of ANSI color formatting improves output readability in terminal environments.

Example outputs generated by the system include random numbers, coin flip results such as "Heads," and humorous responses like a joke about a computer catching a virus. These outputs illustrate the versatility of the agent while maintaining controlled execution.

Key insights from the study indicate that controlled execution mechanisms are essential for preventing unsafe operations. Additionally, the modular design of the system simplifies future extensions, and interactive capabilities enhance overall user experience and engagement.

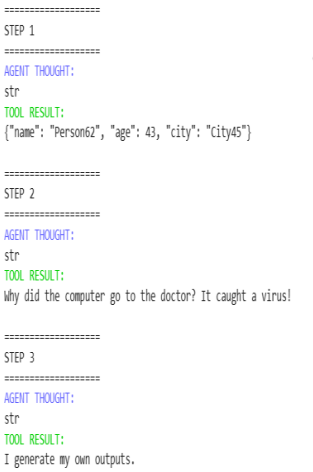

Figure 1.

Figure 2.

V.CONCLUSION

This paper presented a controlled autonomous agent system that utilizes predefined Python execution tools to ensure safe and reliable operation. Two models were developed and analyzed: a deterministic agent and an interactive agent. The findings demonstrate that restricting execution to predefined commands significantly improves system safety while maintaining functional capability.

Furthermore, the tool-based architecture enhances modularity, making the system easier to extend and maintain. The interactive agent, in particular, provides improved usability by allowing dynamic user input while still operating within safe boundaries. Future work may focus on integrating natural language understanding, enabling multi-tool orchestration, and incorporating learning-based decision-making to further enhance the capabilities of autonomous agents.

VI. ACKNOWLEDGEMENT

The authors would like to express their sincere gratitude to the academic community and open-source contributors for their valuable resources and support in the field of artificial intelligence and autonomous systems. Their contributions have played a significant role in the development of this work.

VII.REFERENCES

1. Russell, S., Russell, S., and Norvig, P.,Artificial intelligence:AModern Approach
2. Goodfellow, I., Bengio, Y., Courville, A.Deep Learning
3. Python Software Foundation. Python Documentation
4. OpenAI Research on Tool-Using Agents