UpdAgent: AI Agent Version Control Framework for Real-time Updation of Tools

Praneeth Vadlapati
University of Arizona,
Tucson, USA
praneethv@arizona.edu,
ORCID: 0009-0006-2592-2564

Abstract: Due to the growing usage of agents in artificial intelligence (AI), the versioning of the tools has become essential. New tools or their versions are introduced regularly. Older versions eventually become obsolete. Servers often face maintenance downtime, which obstructs APIs from being used. Version management is common in software development. Similarly, this paper introduces UpdAgent, a version control system to manage the versions of tools used by AI-driven applications that are based on agents and Large Language Models (LLMs). This centralized management system allows the tool providers to deliver real-time updates to add or improve functionalities, resolve issues in existing tools, and immediately revert updates that generate new bugs or errors. The system is designed to streamline real-time updates to tool functions and APIs, ensuring that the LLMs utilize new or updated tool functionalities and avoid APIs that are under maintenance or obsolete. Automated testing is performed to automate the avoidance of tools that produce errors. The management results in automated adaptation to updated tools and reduced delays in the AI-based applications to enhance system reliability. The experiment was successful in setting up the tables and updating the tools using new data. The code is available at github.com/Pro-GenAI/UpdAgent.

Keywords: Artificial Intelligence (AI), AI agents, agentic framework, version control

I. Introduction

Agents in AI are programs or systems that are capable of performing required tasks on their own by designing their own workflow and utilizing the available tools [1], [2]. Agents are commonly included in AI-driven systems to expand their capabilities [1], [3], [4]. There is increasing research and utilization of AI agents [5], [6], [7], which utilize tools to automate numerous tasks efficiently. Agents utilize tools for various use cases such as weather, travel, or restaurant recommendations. Tools might utilize a source code or an API URL, both of which face frequent updates that might include feature releases and critical security updates. The updates include feature introductions, resolutions of bugs and errors, or removal of unused and unmaintained features. Updates are crucial for maintaining performance and usability [8], [9], [10]. A lack of updates could lead to disruptions in LLM-agent interactions, potentially degrading the experience of the end users. Functionality updates are frequent for software [11] such as tools, and a regular manual updation of the tools consumes an enormous amount of time for development and testing.

A. Proposed system and its benefits

Structured automated updates from the providers could ensure continuous error-free functionality of applications [12], including AI-based applications, and eradicate the need for manual updation and testing. During the downtime of an API-based tool, an alternative tool could

be used immediately to ensure the uninterrupted functionality of AI-based applications. This paper proposes UpdAgent, a centralized version management framework designed to solve the challenges with various updates to the tools. Tools are systematically cataloged in the system with relevant details in a structured form. During the update of versions, the details of the tool can be updated according to the new version. The method additionally handles the changes in API keys that might be updated due to expiry or security reasons.

B. Related Work

Prior efforts exist in agent-based architectures that focus on the integration of agents with LLM-based applications [13], [14], [15] and API-based agents in agentic AI environments [16]. Existing systems consider the dynamic nature of APIs in the current digital landscape [17], [18]. Current research on version management focuses on source codes [19], [20] and leaves a gap in the centralized management of the different versions of numerous agent tools. Existing LLM frameworks such as LangChain provide agent orchestration [21] but lack version management and automated testing to manage dynamic updates. UpdAgent addresses dynamic updates to agents using a centralized system upon creation or updation of agent tools and integrates real-time validation upon creation or updation.

II. METHODS

A. Creating a tool table

A tool table is created to serve as a central table for tool details such as tool ID, tool name, description, status, version, release notes, timestamp of the last update or creation, API URL (if any), request method, API key, payload description, example values, and source code (if any). The tool ID is unique to each row. The description of a tool allows LLMs to select only the latest available tool for each task. The table is structured to contain the required information about a tool in a structured way, allowing the system to use a tool by utilizing the latest configurations and capabilities.

B. Creating a version log table

To manage version history and record changes, all tool versions are stored in a version archive table. On new updates to an agent, a copy of its data is saved, allowing the system to maintain a complete history of changes and revert if necessary. The logs of version history support developer activities [22], [23], such as tracking change history, debugging, troubleshooting the system, user feedback correlation of the complaints, correlation of error logs with version logs, auditing, and source code package dependency management.

C. Creation of tools

Creating a new tool involves the creator incorporating the tool data into the UpdAgent system. This step allows developers to add new functionalities into the agentic system for consideration and utilization by the LLM. The required tool details are provided during the experiment and must be provided for the creation. The system records the tool data in the tool table and the archival data in the version log table. This process enables the expansion of the toolset to enhance the capabilities of agentic systems. The storage of each tool in the version log table allows unaltered versions for future reference, which allows the system to maintain records of all tools from their introduction to the system to offer comprehensive logs of the versions.

D. Updation and archival of tools

Updations of tools are the crucial components of the system to modify existing tool details and store a new version in both tables. Changes may include altering any value other than the tool ID. In this method, the details of a tool in the tool table are updated with new details to ensure the AI-driven system accesses the latest details of only active tools. A tool can be marked as down for maintenance by changing the status to "Maintenance." Archival of a tool is performed by changing the status to "Archived." Tools that possess no active status are saved in the version log table and not in the tool table. The archival of a tool is essential in cases of unavailability, deprecation, or replacement by a provider with a better alternative tool.

```
{
        'toolID': 'weathr1',
        'tool_name': 'Weather API',
        'tool_desc': 'Provides weather information',
        'status': 'Active',
        'version': 'v1.1.0',
        'updated_at': '2024-07-01 00:00:00',
        'URL': 'example.com/v1.1/weather',
        'request_method': 'GET',
        'API_key': '<API key here>',
        'payload_desc': {
            'location': 'should be a string in format "city,
country"' },
        'sample_values': { 'location': 'New York, US' },
        'release_notes': 'Added support for multiple languages
and updated API function',
        'python_function': '"
        import requests  # Added import statement
        def get_weather() -> str:
                location = '{location}'
                weather_data = {{ 'New York, US': 'Sunny',
                                'Paris, France': 'Cloudy' }}
                if location in weather_data:
                        return weather_data[location]
                else:
                        raise ValueError('Location not found')
        get_weather()
        '"
}
```

Fig. 1. Data to update a tool

E. Automated testing of a tool

Automated testing of a new or updated tool is performed as soon as a tool is created or updated in the database. The sample value added during the creation of a tool is utilized to test the tool automatically. This facilitates the identification of various issues in the source code that arise due to differences in the architectures, differences in software or package versions, or lack of installation of a package in the system environment. This allows early detection and mitigation of errors with APIs or source code at the time of creation or updation of a tool. This process allows for testing the compatibility and functionality of a tool within the system and verifying whether the tool performs as expected. The early testing approach reduces the risk of runtime errors that arise when an end-user expects a response and waits for it. Automated alerts about errors could

allow tool creators and system developers to resolve configuration issues, improve overall system reliability, and ensure that the tools function as expected, which is crucial when deployed in a large AI-driven application.

III. Results

A. Creating tool table and storing data

An initial setup of the tool table allowed a structured storage of tool information that includes tool ID, description, status, and more columns. This setup facilitated rapid access to the latest details of each tool, enabling LLMs to interact with the most current version of any tool seamlessly.

TABLE I. SUMMARY OF THE TOOL TABLE STRUCTURE

tool ID	tool name	tool_desc	status	version	URL	payload_desc
weathr1	Weather API	Provides weather information	Active	v1.0.0	example.com/ weather	{'location': 'should be a string in format "city, country"'}
rstrnt2	Restaurant API	Provides restaurant information	Active	v1.0.0	example.com/ restaurants	{'cuisine': 'should be a string', 'location': 'should be a string in format "city, country"'}
travel3	Travel API	Provides travel information	Maintenance	v1.0.0	example.com/ travel	{'origin': 'should be a string in format "city, country"', 'destination': 'should be a string in format "city, country"'}

B. Creating version log table and storing data

Establishing a version log with the complete history of the versions has established an efficient mechanism for tracking changes to each tool throughout its life cycle. Each update to a tool is stored as a separate entry in this archive, which allows the system to maintain a record of tool versions. The structure of the version log table is initially the same as the structure of the tool table, as mentioned above.

C. Updating the version of a tool

The ability to update tool versions has been tested by updating existing tool details with new data. The changes were reflected in the tool table, while the historical data with the details of both the new and old versions were preserved and archived successfully in the version log table. This process allowed the system to maintain the latest configurations in real-time while preserving the complete update history of each tool.

TABLE II. UPDATED ROW IN TOOL TABLE

tool ID	tool name	tool_desc	status	version	URL	payload_desc
weathr1	Weather API	Provides weather information	Active	v1.1.0	example.com/ v1.1/weather	{'location': 'should be a string in format "city, country"'}

TABLE III. UPDATED STRUCTURE OF VERSION LOG TABLE

tool ID	tool name	tool_desc	status	version	URL	payload_desc
weathr1	Weather API	Provides weather information	Active	v1.0.0	example.com/ weather	{'location': 'should be a string in format "city, country"'}
rstrnt2	Restaurant API	Provides restaurant information	Active	v1.0.0	example.com/ restaurants	{'cuisine': 'should be a string', 'location': 'should be a string in format "city, country"'}
travel3	Travel API	Provides travel information	Maintenance	v1.0.0	example.com/ travel	{'origin': 'should be a string in format "city, country"', 'destination': 'should be a string in format "city, country"'}
weathr1	Weather API	Provides weather information	Active	**v1.1.0**	example.com/ v1.1/weather	{'location': 'should be a string in format "city, country"'}

IV. DISCUSSION

The implementation of the UpdAgent framework introduces a structured approach to managing AI agent tool updates similar to how software updates are processed. The centralized tool table empowers LLMs to select the latest versions of tools to utilize for each task. The dual-table approach of the framework is proven to be effective in performing real-time updates while maintaining the records with a history. The automated updates to the tools replace the manual updation process and enhance the overall reliability of the AI-driven applications that utilize the framework. Furthermore, retaining the records of changes supports the transparency of the system to monitor and analyze in the future. However, the scalability of the system remains a concern since a huge number of tools could be created considering the current landscape of research on agents. As the number of agents and the usage of agents surge, future iterations of the system could require optimizations to improve performance.

V. CONCLUSION

UpdAgent offers a robust version control solution for tools used by AI agents in the current LLM environments. This addresses the challenges of regular updates in the current surge in the research, utilization, and demand of AI agents. Additionally, the system stores the current and historical versions for future analytics. The centralized storage of tools and structured version logs support efficient debugging, transparent audits, and flexible tool management. This approach mitigates issues related to the unavailability of APIs or bugs and errors in the existing source code. The centralized database ensures that only the latest available error-free versions of the tools are selected, which ensures uninterrupted responses are maintained. The central database allows transparent and efficient debugging of the system, transparent audits for analytics, and a management active and archived tool. The avoidance of disruptions in the operations of AI-driven systems could create a strong foundation for future scalability and adaptability of agent-driven AI applications. Automated testing allows the system to utilize only error-free tools. Future improvements to the system include analyzing the input data required by APIs to detect whether the API providers collect the personal data of the users and avoid giving away unnecessary data to

the providers. The dual-table approach of the system enables a reliable framework that creates a way for enhanced and resilient agent-based architectures.

REFERENCES

[1] Z. Xi et al., "The Rise and Potential of Large Language Model Based Agents: A Survey," Sep. 19, 2023, arXiv:2309.07864. [Online]. Available: http://arxiv.org/abs/2309.07864

[2] Anna Gutowska, "What are AI agents?," IBM. [Online]. Available: https://www.ibm.com/think/topics/ai-agents

[3] S. Kapoor, B. Stroebl, Z. S. Siegel, N. Nadgir, and A. Narayanan, "AI Agents That Matter," Jul. 01, 2024, arXiv:2407.01502. [Online]. Available: http://arxiv.org/abs/2407.01502

[4] S. Sharma and A. Ahlawat, "Architecture and Types of Intelligent Agent and Uses of Various Technologies," in 2022 3rd International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), 2022, pp. 1–5. doi: 10.1109/ICICT55121.2022.10064524.

[5] A. Chan et al., "Visibility into AI Agents," in Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency, in FAccT '24. New York, NY, USA: Association for Computing Machinery, 2024, pp. 958–973. doi: 10.1145/3630106.3658948.

[6] Q. Huang et al., "Position Paper: Agent AI Towards a Holistic Intelligence," Feb. 28, 2024, arXiv:2403.00833. [Online]. Available: http://arxiv.org/abs/2403.00833

[7] S. Kelly, S.-A. Kaye, and O. Oviedo-Trespalacios, "What factors contribute to the acceptance of artificial intelligence? A systematic review," Telematics and Informatics, vol. 77, p. 101925, Feb. 2023, doi: 10.1016/j.tele.2022.101925.

[8] A. Mathur, N. Malkin, M. Harbach, E. Peer, and S. Egelman, "Quantifying Users' Beliefs about Software Updates," in Proceedings 2018 Workshop on Usable Security, Feb. 2018. doi: 10.14722/usec.2018.23036.

[9] P. Rajivan, E. Aharonov-Majar, and C. Gonzalez, "Update now or later? Effects of experience, cost, and risk preference on update decisions," Journal of Cybersecurity, vol. 6, no. 1, p. tyaa002, Mar. 2020, doi: 10.1093/cybsec/tyaa002.

[10] K. Vaniea and Y. Rashidi, "Tales of Software Updates: The process of updating software," in Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, in CHI '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 3215–3226. doi: 10.1145/2858036.2858303.

[11] M. Fleischmann, M. Amirpur, T. Grupp, A. Benlian, and T. Hess, "The role of software updates in information systems continuance — An experimental study from a user perspective," Decision Support Systems, vol. 83, pp. 83–96, Mar. 2016, doi: 10.1016/j.dss.2015.12.010.

[12] F. Angermeir, J. Fischbach, F. Moyón, and D. Mendez, "Towards Automated Continuous Security Compliance," Jul. 31, 2024, arXiv:2407.21494. [Online]. Available: http://arxiv.org/abs/2407.21494

[13] A. Arslan, "Exploring LLM-based Agents: An Architectural Overview," Current Trends in Computer Sciences & Applications, vol. 3, no. 3, pp. 405–411, Jun. 2024, doi: 10.32474/CTCSA.2024.03.000162.

[14] L. Wang et al., "A survey on large language model based autonomous agents," Front. Comput. Sci., vol. 18, no. 6, Mar. 2024, doi: 10.1007/s11704-024-40231-1.

[15] T. Guo et al., "Large Language Model based Multi-Agents: A Survey of Progress and Challenges," Apr. 19, 2024, arXiv:2402.01680.

[16] H. Shen et al., "ShortcutsBench: A Large-Scale Real-world Benchmark for API-based Agents," Jul. 22, 2024, arXiv:2407.00132. [Online]. Available: http://arxiv.org/abs/2407.00132

[17] S. Serbout and C. Pautasso, "An Empirical Study of Web API Versioning Practices," in Web Engineering, I. Garrigós, J. M. Murillo Rodríguez, and M. Wimmer, Eds., Cham: Springer Nature Switzerland, 2023, pp. 303–318. doi: 10.1007/978-3-031-34444-2_22.

[18] R. Sun, Q. Wang, and L. Guo, "Research Towards Key Issues of API Security," in Cyber Security, W. Lu, Y. Zhang, W. Wen, H. Yan, and C. Li, Eds., Singapore: Springer Nature Singapore, 2022, pp. 179–192.

[19] N. N. Zolkifli, A. Ngah, and A. Deraman, "Version Control System: A Review," Procedia Computer Science, vol. 135, pp. 408–415, Jan. 2018, doi: 10.1016/j.procs.2018.08.191.

[20] N. Deepa, B. Prabadevi, L. B. Krithika, and B. Deepa, "An analysis on Version Control Systems," in 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), 2020, pp. 1–9. doi: 10.1109/ic-ETITE47903.2020.39.

[21] S. Rasal and E. J. Hauer, "Navigating Complexity: Orchestrated Problem Solving with Multi-Agent LLMs," Jul. 10, 2024, arXiv:2402.16713. [Online]. Available: http://arxiv.org/abs/2402.16713

[22] S. Gu, G. Rong, H. Zhang, and H. Shen, "Logging Practices in Software Engineering: A Systematic Mapping Study," IEEE Transactions on Software Engineering, vol. 49, no. 2, pp. 902–923, 2023, doi: 10.1109/TSE.2022.3166924.

[23] J. Cândido, M. Aniche, and A. van Deursen, "Log-based software monitoring: a systematic mapping study," PeerJ Computer Science, vol. 7, p. e489, 2021, doi: 10.7717/peerj-cs.489.