AI Agent Standardization using an Agent Package Manager

AUM Shankar Javalgikar[1], AMEY Shankar Chavan[2], Mahesh Shivaji Kadam[3] and Prof.Monali Bansode[4]

[1234]*Computer Department, Savitribai Phule Pune University, Pune, Maharashtra, India.*

Abstract*—As AI systems grow in complexity, the need for reusable, modular AI components has become essential. However, the field lacks standardization, with AI agents and tools developed across diverse frameworks, each with unique dependency structures, communication protocols, and functionality. We propose a system that standardizes the development, sharing, and reuse of AI agents and components, aiming to streamline AI development and foster ecosystem interoperability. By introducing a centralized package manager, this framework allows developers to package agents with defined input-output schemas, dependencies, and versioning information, enabling seamless integration across different applications and environments. Key features include a dependency resolution engine that addresses conflicts between frameworks, semantic versioning for backward compatibility, and a registry for discovering and installing agents. This system promotes modularity, reduces development time, and enhances collaboration by allowing components to be readily reused, configured, and deployed. Through standardization, the system lays the groundwork for scalable, multi-agent workflows and democratizes access to sophisticated AI capabilities, empowering developers and organizations to leverage AI with greater efficiency and flexibility. This paper presents the theoretical framework, benefits, and anticipated challenges of the proposed package manager, underscoring its potential to transform AI system design through an AI agent package manager.

***Index Terms*—Agents, Packages, Components, Standardization**

I. INTRODUCTION

Recent advancements in Large Language Models have sparked a movement in software engineering, focusing on AI agents that utilize natural language processing to solve complex problems. There are numerous repositories on GitHub for AI agents, each with its own incompatible abstractions. On average, it takes significant developer time to understand and integrate these implementations.

Additionally, the tight coupling of workflows and data within agents hinders interoperability. Executing these systems often requires resolving dependencies and running complex setups. This project proposes a standardized framework for implementing AI agents using a modular approach, akin to npm, to facilitate easy isolation and integration of agent components.

By introducing a standardized framework for AI agents, this project seeks to streamline the integration, reuse, and interoperability of AI components. The proposed system will function as a package manager for AI agents, enabling developers to package, version, and share agents in a modular format, similar to how npm transformed JavaScript development. This approach addresses the challenges of dependency management, version compatibility, and cross-framework functionality, making it significantly easier for developers to discover, integrate, and utilize diverse AI agents in their projects.

Key features of the framework include automatic dependency resolution, standardized input-output interfaces, and a central registry for publishing and retrieving AI agents. By encapsulating agents with defined dependencies and standardized configurations, the framework reduces the setup complexity, allowing developers to focus on higher-level tasks such as customizing agents or integrating them into larger AI workflows. Additionally, this modular approach enables a plug-and-play experience, where developers can quickly experiment with and substitute different agents, fostering rapid innovation and experimentation.

II. LITERATURE REVIEW

Despite their potential, existing implementations of these agents are often fragmented across various repositories and frameworks, each with unique abstractions and dependencies that complicate integration. This lack of standardization highlights the need for a modular, interoperable framework that can streamline AI agent development, similar to the role of package managers in other software ecosystems.

The studies below were reviewed before proposing this system:

Paper [1] highlights critical security and ethical considerations in deploying AI agents capable of code generation, particularly the risks associated with executing LM-generated code on personal devices, which could result in unintended harmful actions. To address these issues, Docker is used for safer execution environments.
Paper [2] highlights the growing capability of LLM-based agents to autonomously handle software engineering tasks, from bug reproduction to patch generation, underscoring the need for robust context retrieval and dependency management. This approach aligns with our project's focus on creating a standardized, modular framework that allows AI agents to integrate easily while managing dependencies and preserving interoperability.

Paper [3] research on AI agent evaluations underscores the importance of standardized, cost-controlled benchmarking for assessing AI agents across diverse tasks, an area our project also addresses through modular, interoperable agent packaging. Similar to their focus on optimizing for cost and reproducibility, our framework aims to provide consistency in agent deployment and evaluation by managing dependencies and ensuring compatibility across setups. Additionally, as our project scales, we recognize the need to evaluate not only computational costs but also the broader impact of AI agents, aligning with the study's call for holistic cost assessments that include environmental and maintenance considerations.

III. METHODOLOGY

The core components of the proposed AI agent package manager draw significant inspiration from the structure and functionality of npm, the package manager for JavaScript, which revolutionized how developers share and reuse software modules. Like npm, this AI package manager would have a central registry, dependency resolver, version manager, and command-line interface (CLI), each serving to standardize and simplify the process of packaging, sharing, and integrating AI agents.

The central registry serves as a repository where all AI agents are stored, categorized, and made accessible for developers to search, download, and integrate into their projects. Much like npm's registry, this component would provide a single, reliable source for users to access various AI agents with detailed metadata, usage information, and standardized input/output schemas. This registry is vital for fostering a collaborative ecosystem, as it allows developers to contribute their own agents while ensuring all packages follow a common format, ensuring consistency and ease of integration.

The dependency resolver is another essential component that manages the intricate web of dependencies AI agents often require, similar to npm's package dependency tree. Unlike traditional software packages, AI agents may depend on specific machine learning models, data preprocessing steps, or even other agents to function correctly. The resolver would analyze these dependencies, identify potential version conflicts, and ensure that all required packages are compatible and installed in the correct order. This component is crucial to prevent the "dependency hell" often encountered when multiple AI agents or frameworks are used together, by automatically managing compatible versions and resolving conflicts.

Next, the version manager enforces semantic versioning (semver) standards across AI agents, similar to npm's approach with modules. Semantic versioning enables developers to specify which versions of an agent are compatible with their application, marking major, minor, and patch updates in a way that is consistent and predictable. The version manager would also generate lock files (e.g., agent-lock.json), similar to npm's package-lock.json, ensuring that the same version of dependencies is used across different environments, thus guaranteeing reproducibility and consistency in deployments. This component ensures that updates to an AI agent do not inadvertently break compatibility with other agents or systems that rely on it.

The command-line interface (CLI) is the primary interaction point for users, offering commands for common tasks such as searching, installing, updating, and removing agents. Drawing directly from npm's model, the CLI would allow users to manage agents with simple commands like agent install, agent update, and agent remove, streamlining the workflow for integrating AI agents into projects. Additionally, the CLI would provide options to list installed agents, view available versions, and inspect dependencies, giving developers a transparent view of their environment and dependencies.

Together, these core components work to achieve a streamlined, modular system for managing AI agents, allowing for easy sharing, reuse, and interoperability across diverse AI projects. Like npm, which abstracted complex module management into a user-friendly interface, the AI package manager would encapsulate the complexities of AI dependencies and compatibility management into a cohesive, developer-friendly system. This structure not only enhances productivity by reducing the setup and configuration time for AI projects but also encourages best practices in modularity and version control, fostering a collaborative AI ecosystem that is greater than the sum of its parts.

The centralized registry is a pivotal component of the proposed AI package manager, functioning much like npm's registry by serving as the single, reliable repository where all AI agents are stored, indexed, and made accessible for the community. This centralized repository plays a foundational role in organizing and categorizing AI agents, allowing developers to easily find, share, and integrate AI modules across projects. The registry's design focuses on accessibility, reliability, and security, ensuring that every agent available is vetted, versioned, and readily usable in diverse AI workflows.

At its core, the centralized registry is designed to support a diverse array of metadata fields that define each AI agent in detail. This metadata includes the agent's name, description, version, dependencies, input/output schemas, and compatibility with specific frameworks (e.g., PyTorch, TensorFlow). Such information is crucial, as it provides developers with the necessary details to assess compatibility and understand the agent's intended functionality without deep technical exploration. The registry's metadata design mimics npm's package.json in structure, providing a standardized format that ensures consistency across all published agents, so developers can rely on familiar structures for quick comprehension and integration.

IV. RESULTS AND DISCUSSION

This paper discusses a package manager that addresses the current fragmentation and integration challenges in the AI agent ecosystem. By drawing parallels with npm, the proposed AI package manager establishes a foundational architecture for modularity, reusability, and interoperability among AI agents. Below, we discuss the anticipated impact, benefits, and potential challenges of this system, as well as future research directions.

A. Impact on AI Development Workflow

The AI package manager is expected to significantly streamline the development workflow for AI agents by enabling modular, reusable, and easy-to-integrate components. Much like npm revolutionized web development by simplifying the sharing and integration of JavaScript modules, this package manager aims to create a standardized approach to packaging, versioning, and deploying AI agents. Developers can expect substantial time savings, as they will no longer need to spend extensive time understanding and manually configuring dependencies across various frameworks. By providing a structured environment with dependency management, version control, and standardized interfaces, the proposed system can reduce integration complexity and foster a more collaborative AI development ecosystem.

Benefits

1. Modularity and Reusability: The package manager allows developers to package agents in a modular format, promoting reusability across different projects. This modularity encourages a plug-and-play approach, where agents can be used interchangeably and in various configurations, speeding up experimentation and innovation.
2. Improved Dependency and Version Management: With built-in dependency resolution and semantic versioning, the package manager simplifies managing complex dependency trees, avoiding conflicts that traditionally lead to "dependency hell." This aspect will be particularly beneficial in multi-agent systems, where compatibility across different versions and frameworks can be challenging.
3. Enhanced Interoperability: By enforcing standardized input/output schemas and providing framework-agnostic support, the package manager will improve interoperability between agents built on different frameworks, making it easier for developers to build cross-framework AI workflows and systems.
4. Community-Driven Ecosystem: A centralized registry and guidelines for publishing agents will encourage community collaboration, allowing developers to share their agents, benefit from

community contributions, and collectively enhance the AI ecosystem. This approach mirrors npm's open-source success, potentially fostering a large repository of ready-to-use AI agents for a variety of applications.

Parameter	AI Agent Package Manager	Existing Solutions (e.g., LangChain, AutoGen)
Modularity	High modularity with standardized packaging format, supporting plug-and-play agent integration	Often limited modularity, with agents tightly coupled to frameworks or specific setups
Interoperability	Framework-agnostic with standardized I/O schemas for easy cross-framework integration	Typically limited to specific frameworks; requires custom integration for interoperability
Dependency Management	Automated dependency resolution and version control using semantic versioning	Dependency management varies, often manual or minimally supported, leading to potential conflicts
Centralized Registry	Single, organized registry for publishing, searching, and retrieving agents	No centralized repository; agents are spread across individual repositories (e.g., GitHub)
Community Collaboration	Supports community-driven contributions with guidelines, categorization, and clear governance	Community contributions are often ad hoc, with no formalized guidelines for quality control

V. CONCLUSION

The proposed AI Agent Package Manager aims to transform the fragmented AI ecosystem by introducing a standardized, modular approach to packaging, sharing, and integrating AI agents, much like npm did for JavaScript. By enabling automated dependency management, version control, and framework-agnostic interoperability, the package manager simplifies AI development, enhances reusability, and fosters community-driven innovation. While challenges in security, scalability, and framework compatibility remain, this framework has the potential to significantly improve the efficiency and accessibility of AI agent development, laying a foundation for more collaborative and scalable AI systems. Future work will focus on implementation and testing to bring this concept into practice, enabling a reliable, robust ecosystem for AI agents.

ACKNOWLEDGMENT

We would like to express our sincere gratitude to the faculty and staff of the Sinhgad Institute of Technology, Lonavala, Department of Computer Engineering, for their invaluable support and encouragement throughout this project. Their guidance and resources were instrumental in bringing this work to completion. Additionally, we extend our heartfelt thanks to our project guide, Mrs. Monali Bansode, whose insightful direction and unwavering support guided us through each stage of this research. This project would not have been possible without their dedication and assistance.

REFERENCES

[1] J. Yang, C. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press, "SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering," Princeton University, Princeton Language and Intelligence. Available at: {jy1682, carlosej, awettig, kl5675, shunyuy, karthikn, ofirp}@princeton.edu.

[2] Y. Zhang, H. Ruan, Z. Fan, and A. Roychoudhury, "AutoCodeRover: Autonomous Program Improvement," National University of Singapore. Available at: {yuntong, hruan, zhiyufan, abhik}@comp.nus.edu.sg.

[3] S. Kapoor, B. Stroebl, Z. S. Siegel, N. Nadgir, and A. Narayanan, "AI Agents That Matter," Machine Learning (cs.LG); Artificial Intelligence (cs.AI), arXiv:2407.01502, 2024.